MOPS®

Mathematical OPtimization System

MOPS

Mathematical OPtimization System

User manual

MOPS Version 9.x
16.07.2008

MOPS Optimierungssysteme GmbH & Co. KG
Alfener Weg 20
D-33100 Paderborn
Telefon: +49 (0)30 838 55009, +49 (0)5251 66149
E-mail: uwe.suhl@mops-optimizer.com
http://www.mops-optimizer.com

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

MOPS® is a registered trade mark (Deutsches Patentamt Nr. 2017758). All rights reserved. No parts of this work may be reproduced in any form or by any means - graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems - without the written permission of the publisher.

Products that are referred to in this document may be either trademarks and/or registered trademarks of the respective owners. The publisher and the author make no claim to these trademarks.

While every precaution has been taken in the preparation of this document, the publisher and the author assume no responsibility for errors or omissions, or for damages resulting from the use of information contained in this document or from the use of programs and source code that may accompany it. In no event shall the publisher and the author be liable for any loss of profit or any other commercial damage caused or alleged to have been caused directly or indirectly by this document.

Printed: Juli 2008 in Berlin

Table of Contents

Chapter I Introduction 2

1 Introduction ... 2
2 What's new in MOPS 9.x ... 3

Chapter II MOPS.DLL 6

1 Using MOPS.DLL ... 6
2 Avoiding errors ... 7
3 Basic functions ... 8
Basic functions ... 8
4 Extended functions ... 22
Extended functions ... 22
5 Embedding the DLL ... 42
Declarations ... 42
Visual Basic 6.0 ... 43
Visual Basic .NET ... 44
Visual C/C++ 6.0 ... 46
C# ... 47
6 64 Bit Version ... 49
64 Bit Version ... 49

Chapter III MOPS.LIB 51

1 Using MOPS.LIB ... 51
2 MOPS.LIB Functions ... 51
3 Calling MOPS.LIB from C/C++ ... 54

Chapter IV MOPS.EXE 56

1 Using MOPS.EXE ... 56

Chapter V MOPS STUDIO 58

1 Using MOPS STUDIO ... 58

Chapter VI MOPS Parameter 66

1 MOPS Parameter ... 66
2 Input Parameter ... 66
Input Parameter ... 66
File names ... 66
LP Parameters ... 69
MPS vector names ... 77
Ressource limits ... 79
Tolerances ... 84

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

Other Parameters ... 89
IP Parameters ... 95
3 Output Parameters ... 113
Output Parameters ... 113
Error Message ... 118
Chapter VII MOPS Input Files 124
1 MOPS Input Files ... 124
2 Profiles ... 124
3 Data Input ... 125
Data Input ... 125
MPS Files ... 125
MPS Files ... 125
Example ... 126
Sections ... 127
Integer Variables ... 129
Triplet Files ... 130
Triplet Format ... 130
Example ... 131
Extended Integer Types ... 132
Extended Integer Types ... 132
Enhanced MPS Format ... 133
Enhanced Triplet Format ... 136
Chapter VIII MOPS Output Files 138
1 MOPS Output Files ... 138
2 Solution Files ... 138
3 Log Files ... 139
4 fort.xxx Files ... 139
Chapter IX Case Burma 142
1 Problem ... 142
2 Solution ... 143
3 MPS File ... 145
4 Triplet File ... 146
5 AMPL ... 148
Chapter X References 151
1 References ... 151
Index 152

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

Chapter 1

Introduction

1 Introduction

1.1 Introduction

MOPS (**M**athematical **OP**timization **S**ystem) is a high performance solver for large LP and IP models.

The abbreviation ***LP*** stands for "Linear Programming", which describes a class of optimization models where all relations are linear and only continuous variables are allowed. ***IP*** stands for "Integer Programming", which means pure integer or mixed models with integer and continuous variables. The mathematical formulation for these model classes is:

Maximize / Minimize $\mathbf{c'x}$
$\mathbf{rl} \leq \mathbf{A\,x} \leq \mathbf{ru}$
$\mathbf{l} \leq \mathbf{x} \leq \mathbf{u}$
$\mathbf{x_j}$ **integer for** $\mathbf{j} \in \mathbf{Jl} \subseteq \mathbf{J}$

where $J = \{1,\ldots,n\}$, $x, c, l, u \in R^n$, $rl, ru \in R^m$ and A is a real (m,n)-matrix. The vectors l, u, rl, ru may have elements representing plus or minus infinity. This formulation includes free variables and ranged constraints. This representation is called the *external model representation* (EMR) in contrast to the *internal model representation* (IMR), which is the way a model is mapped into the MOPS internal data structures.

MOPS V 8.x supports the following integer variable types:

Name	Definition
Continuous variable	can take on any real value in a specified range
Binary variable	can take on only the values zero or one
Integer variable	can take on only integer values in a specified range.
Semi-continuous variable (SC-variable)	This variable is either zero or can take on any real value in a specified positive range. It can also have a fixed charge cost K associated with it if the variable is nonzero.
Semi-integer variable (SI-variable)	This variable is either zero or has an integer value in a specified positive range. It can also have a fixed charge cost K associated with it if the variable is nonzero.
Partial-integer variables (PI-variable)	An integer variable, which can only take integer values in a small user defined range. Outside this range the variable can take on any real value in a specified range.
Special Ordered Set of type 1 (SOS1)	A set of variables, where at most one variable of this set may be nonzero. These variables are not necessarily integer variables.
Special Ordered Set of type 2 (SOS2)	A set of continuous variables, where at most two variables of this set may be nonzero and they have to be adjacent.
Special Ordered Set of type 3 (SOS3)	A set of 0-1-variables which sum up to 1.
LI-variables	A set of 0-1-variables to linearize a separable function of nonlinear continuous variables. The formulation of LI-variables is an alternative modeling technique of SOS2.

The input format of the extended integer types in the new branch-and-cut code is described in Extended Integer Types.

The MOPS libraries are primarily designed for use under Windows operating systems, but versions for other platforms are also available. Under Windows the following interfaces are supported:

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

- Dynamic Linked Library (MOPS.DLL, 32 and 64 bit)
- Static Library (MOPS.LIB, 32 and 64 bit)
- Command Line Executable (MOPS.EXE and MOPS64.EXE)
- MOPS Studio
- Clip-MOPS (Excel Plug-In)

These interfaces offer a variety of functions for generating, loading, modifying, saving and optimizing LP and IP models.

There is a MOPS White Paper which can be used in conjunction with this user manual. The MOPS White Paper offers background to this user manual and covers:

- history of the MOPS system,
- basic algorithms and concepts,
- system and its architecture,
- research and development aspects,
- references (books and research papers).

1.2 What's new in MOPS 9.x

- MOPS 9.x has an improved memory management. The required amount of main memory is allocated as a function of problem size and the parameter settings. The parameter xmreal (default is xmreal = 0) determines if memory is allocated automatically or if xmreal > 0 if xmreal MB are to be allocated by request.
- MOPS 9.x uses a cut pool to store all derived cuts. If the parameter xusepl is greater than zero (default is xusepl = 0) the cut pool is used otherwise all violated cuts are inserted into the coefficient matrix at each supernode of IP-preprocessing.
- Interior Point and Dual Simplex algorithm for solving LP models are significantly improved. See parameter *xlptyp* for details. The best ordering heuristic to be used during the Interior Point Engine - which is critically for its performance - is determined automatically. See also the parameter xoripm.
- Another outstanding feature of the Interior Point Engine is that it can use multiprocessing. On the Intel Core Duo architecture a 40% speed improvement can be obtained by specifying xnproc = 2 (number of processors to be used).
- A new Branch and Bound / Cut algorithm with extended integer types is available. See parameter *xnewbb*. The new algorithms include different heuristic (*xheutp*), branching (*xbrheu*) and node selection (*xnodse*) strategies. Additionally local search strategies (*xlocs*) can be executed. This algorithm is the default and leads to large performance gains on many integer optimization problems. The old branch-and-bound-algorithm can be used by setting xnewbb = 0.
- The simplex engines and the crossover algorithm of the IPM engine use a newly implemented LU-factorization and a new LU-update which use memory more efficiently. The old LU-factorization and the old LU-update are possible. See the parameters xnwluf and xnwluu .
- If the model contains range constraints, the IP-Preprocessing can be more efficient in many cases, if

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

these constraints are split into a pair of constraints (parameter *xranha*).

- An extended bound reduction technique is used during the IP-preprocessing. This technique sometimes leads to shorter runtimes for some integer models. (parameter *xbndrd*).
- A new heuristic for the initial solution of an IP model is available (when setting parameters *xnodse* = *-3*, *xheutp* = *4* and setting parameter *xmnheu* for the number of nodes in the heuristic).
- Saving and restoring of Branch and Bound trees is no longer supported.
- Improved Gomory Mixed Integer Cuts may reduce solution times for some models.
- Mixed-Integer rounding (xmirct) and flow (path) cuts (xflwct) result in significant performance gains on some integer optimization problems.
- A new DLL-function *Optimize* combines and replaces *OptimizeLP* and *OptimizeIP*.
- The status of an IP solution returned by *GetIPSolution* is now identical to the status returned by *Optimize*.
- Modified return codes of DLL function *MOPS*.
- Completely reworked IMR interface for when using the static MOPS.LIB library from Fortran or C.
- Additional DLL functions for building and modifying models.

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

Chapter II

MOPS.DLL

2 MOPS.DLL

2.1 Using MOPS.DLL

The Dynamic Link Library MOPS.DLL can be used under Windows from a variety of programming languages. Any language that supports 32-bit DLL calls and features the data types listed below can call the MOPS.DLL. The MOPS.DLL can be used from Visual Basic, C/C++, Fortran, Delphi as well as from any .NET language such as VB.NET, C#, Delphi.NET etc. There is also a 64-bit version of the MOPS.DLL

The most recent versions of MOPS.DLL provide besides the Basic Functions a set of Extended Functions, that make it easier to set, retrieve and modify single items of a model, like costs, dual values, right hand sides etc.

Steps for optimization with MOPS.DLL

The following flowchart shows the necessary steps when building and optimizing a model. We use Visual Basic Syntax and assume that the model is transferred via *PutModel()* to the MOPS.DLL:

1. **Data extraction and model generation**
   Primary data is extracted from a database or other sources and modified according to the given problem. A model generator (in your favorite programming language) stores the model data in the following arrays *ia, ja, a, lb, ub, c, typ. S*ee *PutModel()* for a description of the arrays.

2. **Initializing a new model to defaults**
   ```
   ' A new feature since MOPS v9.x is the function Initialize(), which replaces the former functions AllocateMemory() and
   ' InitModel() and which allocates memory automatically.
   rc = Initialize()
   If rc <> 0 then exit
   ```

3. **Setting of non default parameters**
   ```
   ' Use IPM engine for inititial LP and do not use IP-heuristic before the B&C
   rc = SetPameter (" xlptyp=4  xheutp = 0 ")
   If rc <> 0 then exit
   ```

4. **Transferring model to the DLL**
   ```
   rc = PutModel (intyp, inf, m, n, nz, ia(1), ja(1), a(1), lb(1), ub(1), c(1), typ(1))
   If rc <> 0 Then exit
   ```

5. **Optimize model (LP and IP)**
   ```
   rc = Optimize(-1, lstatus, lphase, dfunct)
   ' check return code and status if optimization was successful; -1 maximization, 1 minimization
   ' lphase = 0: LP phase, lphase = 1: IP-phase; lstatus = 0: optimal solution
   ```

6. **Get LP solution or IP solution if optimal or time limit reached with an integer solution**
   ```
   if lphase = 0 and lstatus = 0 then
     'LP-model: optimal LP-solution found - store it in user arrays
      rc = GetLPSolution(xlpsta, xfunct, xs(1), dj(1), stat(1))
   else if lphase = 1 and (lstatus = 0 or lstatus = 3) then
     'optimal IP-solution or feasible IP-solution found (time limit) - store it in user arrays
      rc = GetIPSolution(ipsta, xzbest, xs(1), dj(1), stat(1))
   else
      'error or time limit  and no integer solution
   ```

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

```
end if
If rc <> 0 Then exit
```

As a general principle a function returns an integer return code, which is zero if the call was successful and nonzero in case of an error.

Declarations

To use MOPS.DLL, its exported functions must be declared. See chapter Embedding the DLL for information about how to declare DLL functions.

Model locking

Once the optimization has started, the model will be locked for functions which retrieve model information or which change the model. If you try to use such functions while the model is locked, error code 10 will be returned. After complete optimization, the model will be unlocked and you can for example change parts of it for a new optimization run. This mechanism is necessary, because MOPS transforms great parts of the model, notably during preprocessing. See parameter *ximrrw*, which controls the saving of locked models.

Data types

MOPS.DLL requires in its interface exclusively the following data types. Make sure that your program uses the same types when calling MOPS.DLL functions - if not, errors and crashes might occur.

Name used in this manual	**Fortran**	**C/C++**	**VB**
I4 (integer)	integer*4	long	Long
R8 (floating point)	real*8	double	Double
C*len* (string of length *len*)	character*len	char mystring [len]	String * len

For brevity this manual uses the following variables:

m or *xm*	Number of constraints.
n or *xn*	Number of structural variables.
xj = *xn* + *xm*	Total number of variables (structural + logical).

2.2 Avoiding errors

Errors when passing variables

A correct calling sequence for the MOPS.DLL functions is essential for the stability of our application. Mismatches of declaration or passed variables will result in errors or might even lead to a crash of you development environment, including loss of data.

It is important, that the correct number of variables is passed to a function, that the type of these variables is correct and that the necessary calling conventions are respected. Observe the following hints for a stable application:

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

- Numerical values are passed either as 4-byte integers ("long") or as 8-byte reals ("double").
- Input strings must be zero terminated.
- Output strings must be sufficiently dimensioned. If the MOPS.DLL returns for example a 255 byte long string, than the string passed to the function must also be at least 255 bytes long; the DLL will not extend the string if it is too short.
- Scalar numerical input variables (long, double) are always passed *By Value*.
- Scalar numerical output variables (long, double) are always passed *By Reference*.
- Strings are always passed *By Reference*.
- Arrays are always passed *By Reference*.
- Calling Convention is *STDCALL*

Debugging

All DLL functions pass back a return code, which is zero if the function has been executed successfully, otherwise it is nonzero. Return codes should always be checked to avoid consecutive faults.

If a return code is not zero, in most cases the MOPS parameters *xertyp*, *xline1* and *xline2* provide an error code and a short description of the error. Check these parameters with *GetParameter* for a more precise specification of the occurred error .

When developing your application, you can activate MOPS logging for debugging purposes. The parameter *xoutlv* determines whether a logfile with name *xfnmsg* will be written. Most MOPS functions will write short messages into this logfile when they are called, which it is a very helpful tool in the debugging process. Writing a logfile of course slows down your application, so don't forget to disable logging when running optimizations in productive systems.

Resource limits

A number of MOPS parameters allows you to set resource limits, that lead to a stop or a break of the optimization run when the respective value is reached. There are resource limits for optimization time, quality of the solution, main memory and hard disk space. If you don't know beforehand how the model will behave at run time, you should set resource limits! Particular large models might end up in sheer endless optimization runs, if no resource limits have been set. To stop such a running application you have to force a break, which might lead to a crash and loss of data of the whole development environment (this is especially in Visual Basic 6.0 the case).

2.3 Basic functions

2.3.1 Basic functions

Alphabetical list of MOPS.DLL basic functions

Index

Initialize	Finish	GetIPSolution
GetLPSolution	GetModel	GetParameter
LoadModel	MOPS	Optimize

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

PutModel	ReadMpsFile	ReadProfile
ReadTripletFile	SetParameter	WriteMpsFile
WriteTripletFile		

Discontinued functions

AllocateMemory	FreeMemory	InitModel
OptimizeIP	OptimizeLP	

See *Extended Functions* for more functions.

Finish

Finish () frees all memory previously allocated for the model and closes all open files. This is usually the last MOPS.DLL function you will call in your model generator.

Input

-

Output

Rc (I4)	0: OK <> 0: Error

Example

```
Rc = Finish ()
```

Back to index

GetIPSolution

GetIPSolution (IPStatus, IPFunct, Activity, RedCost, Status) returns the best found integer solution. If a feasible solution has been found the arrays *Activity, RedCost* and *Status* will be filled with the corresponding solution values.

Note, integer variables always get the status 0. Only non integer variables get their original status in the LP-solution.

Input

-

Output

IPStatus (I4)	Status of the IP solution: 0: Optimal (search completed) 1: Optimality not proved
IPFunct (R8)	Objective function value of the IP solution
Activity(1:xj) (R8)	Activities of variables and constraints
RedCost(1:xj) (R8)	Reduced costs of variables and dual values of constraints
Status(1:xj) (I4)	Status of variables in solution:

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

	0: Integer variable, 1: Basic, 2: LB, 3: UB, 4: Fixed, 5: Superbasic, 6: below LB, 7: above UB
Rc (I4)	0: OK <> 0: Error

Example
Solution values for all variables (continuous or integer) > 0 will be written:

```
if xnints > 0 then
      Rc = GetIPSolution (IpStatus, ipFunct, Activity, RedCost, Status)
      do j = 1 to xn
            if dActivity(j) > 0.001 then
                  'Output values ...
            End if
      Enddo
End if
```

Where xnints is the number of integer variables previously returned by MOPS and xn is the number of structural variables.

Back to index

GetLPSolution

GetLPSolution (LPStatus, LPFunct, Activity, RedCost, Status) returns the values of an optimal LP solution. If a solution has been found, the arrays *Activity, RedCost* und *Status* will be filled with the respective values.

Note, if the LP is detected as infeasible during LP-preprocessing the values of the returned solution is meaningless. If you turn off LP-preprocessing (xrduce = 0) and use one of the simplex engines then the values of the array *Status* indicates the infeasible variables.

Input

-

Output

LPStatus (I4)	Status of the LP solution: 0: optimal 1: infeasible 2: unbounded 3: not optimal
LPFunct (R8)	Objective value of the LP solution
Activity(1:xj) (R8)	Activities of variables and constraints
RedCost(1:xj) (R8)	Reduced costs of variables and dual values of constraints
Status(1:xj) (I4)	Status of variables in solution 1: Basic, 2: LB, 3: UB, 4: Fixed, 5: Superbasic, 6: below LB, 7: above UB
Rc (I4)	0: OK <> 0: Error

Example
Return the activities of all basic variables if they are > 0:

```
Rc = GetLPSolution (LpStatus, LpFunct, Activity, RedCost, Status)
```

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

```
If LpStatus= 0 then
      do j = 1 to xj
            if Status(j) = 1 and Activity(j) > 0.001 then
                  ' Output of values
            End if
      Enddo
End if
```

Back to index

GetModel

GetModel (Intype, inf, m, n, nz, ia, ja, a, lb, ub, c, type) returns a complete model to MOPS.DLL. See *PutModel()* for a detailed description of the arrays.

All arrays must be dimensioned sufficiently large. Use *GetDim()* to retrieve the dimensions of the current model in main memory.

Input

Intype (I4)	Mode of storing the model in the returned arrays *ia, ja, a*: 0: Triplets in *ia, ja, a*. The triplets *(i,j,aij)* have no special order. 1: Row-wise storage, where *ia* contains in positions *1,..,m* the start position of the rows in arrays *ja* and *a*. 2: Column-wise storage, where *ja* contains in positions *1,..,n* the start positions of the columns in arrays *ia* und *a*.

Output

inf (R8)	Value to represent "infinity", i.e. any bound that has *inf* as its absolute value is treated as unbounded
m (I4)	Number of rows
n (I4)	number of columns
nz (I4)	Number of nonzero elements
ia(1:nz) (I4)	Row indices (intyp = 0, 2) or start positions of rows (intyp = 1)
ja(1:nz) (I4)	Columns indices (intyp = 0, 1) or start positions of columns (intyp = 2)
a(1:nz) (R8)	Nonzero elements of the matrix (triplets, row-wise or column-wise)
lb(1:n+m) (R8)	Lower bound for structural variables and constraints
ub(1:n+m) (R8)	Upper bound for structural variables and constraints
c(1:n) (R8)	Objective function coefficients
type(1:n) (I4)	Type of structural variable (0: continuous, <>0 integer)
Rc (I4)	0: OK 2: No model present or error in model 10: Model locked

Back to index

GetParameter

GetParameter (Parameter, Value) returns a MOPS parameter. The parameter is always returned as as string. Use*SetParameter()* to change a parameter. See chapter "MOPS Parameters" for a list of the most important parameters.

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

Input

Parameter (C6)	Name of the parameter

Output

Value (C80)	Current value of the parameter
Rc (I4)	0: OK <> 0: Error

Example

```
Rc = GetParameter ("xmxmin", sValue)
```

returns the current value of parameter *xmxmin*.

Back to index

Initialize

Initialize() replaces in MOPS v.9.x the former functions *AllocateMemory()* and *InitModel()*. The function must be called before a model is passed to the MOPS DLL (PutModel, LoadModel, ReadMPSFile, ReadTripletFile). All parameters are set to default values. Parameters can be changed after a call to Initialize by reading a MOPS profile (ReadProfile) or by SetParameter.

Input

-

Output

Rc (I4)	0: OK <> 0: Error

Example

```
Rc = Initialize()
```

Back to index

MOPS

MOPS (FileName) carries out a complete standard optimization run. The passed variable *FileName* contains the file name (optionally with path) of a MOPS profile, in which any MOPS parameter can be set. The function *MOPS()* allocates memory, carries out all initializations, sets parameters specified in the profile, reads the model from an MPS or triplet file, runs LP and IP optimization and writes solutions to files (if *xoutsl* > 0). See chapter "MOPS Parameters" for a description of the most important parameters.

In case of an error, you can use *GetParameter()* to retrieve *xertyp*, *xline1* and *xline2* to get a more specific error description.

Input

FileName (C255)	Name of profile to use (optionally with path, max. 255 characters).

Output

Rc (I4)	0: OK

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

1: Error while reading the profile.
2: Generating default file names caused an error.
3: Bad MOPS parameter (wrong name or value).
4: Error while reading MPS or triplet file.
5: Error in model generator.
6: Error in LP optimization (see *OptimizeLP()* for details).
7: Error while writing solution files.
8: Error in IP optimization (see *OptimizeIP()* for details).

LP-Example
A maximization problem is solved with MOPS parameters taken from *c:\mops\xmops.pro*. After the optimization the return code of the LP run and the objective function value is retrieved:

```
fnpro = "c:\mops\xmops.pro"
Rc = MOPS(fnpro)
Rc = GetParameter("xrtcod", returncode)
If (returncode = 0) Then
      Rc = GetParameter("xlpsta", lpsta)
      If (lpsta = 0) Then
            Rc = GetParameter("xfunct", funct)
      End if
End if
```

IP-Example
The profile contains all relevant optimization parameters. If the profile lies in the current working directory you can only pass the filename of the profile, or else the filename must be preceded by a relative or absolute path. After the optimization return code, IP status and objective function value are retrieved.

```
fnpro = "c:\mops\xmops.pro"
Rc = Mops(fnpro)
Rc = GetParameter("xrtcod", returncode)
Rc = GetParameter("xipsta", lpsta)
Rc = GetParameter("xzbest", zbest)
```

Back to index

LoadModel

LoadModel (Intype, inf, m, n, nz, ia, ja, a, lb, ub, rl, ru, c, type, fcpi) transfers a complete LP/IP model to the MOPS.DLL.Compared to PutModel this function can transfer IP models with extended integer types (SC-, SI-, PI-, LI-variable and SOS of Type 1 and 2) to the DLL. A model with m constraints and n structural variables must be in the following form:

$$\begin{aligned} \min/\max \quad & c' x \\ & bl \le A\,x \le bu \\ & l \le x \le u \end{aligned}$$

with xj being integer for every element j from set JI

The vectors x, c, l, u are real vectors of size *n, A* is a *m x n* matrix of type real, *bl* and *bu* are real *m* -vectors. The vectors *l, u, bl, bu* can have elements, that are plus or minus infinite (+/- inf), to represent unbounded variables or constraints. For equations the values *bl* and *bu* must be identical. *I = {1,..,m}* and *J = {1,..,n}* are index sets for constraints and variables. *JI* is a subset of *J* and contains all indices of integer variables. If *JI* is empty, the model is pure LP.

Note: The convexity constraints of special ordered sets S1-S3 should not be included in the above model. They are generated automatically inside the function LoadModel. However, the size of the output

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

arrays to be passed to the functions GetLPSolution and GetIPSolution have to be increased to n + m + nsos (instead of m+n) where nsos represents the number of special ordered sets. This is so, because the model contains m+ nsos constraints of which only m constraints are generated by the user.

Input

Intyp (I4)	Mode of storing the model in the arrays *ia, ja, a*: 0: Triplets in *ia, ja, a*. The triplets *(i,j,aij)* have no special order. 1: Row-wise storage, where *ia* contains in positions *1,..,m* the start position of the rows in arrays *ja* and *a*. 2: Column-wise storage, where *ja* contains in positions *1,..,n* the start positions of the columns in arrays *ia* und *a*.
inf (R8)	Value that represents infinity (e.g. 1.0E20)
m (I4)	Number of rows
n (I4)	Number of columns
nz (I4)	Number of nonzeros
ia(1:nz) (I4)	Row indices (intyp = 0, 2) or start positions of rows (intyp = 1)
ja(1:nz) (I4)	Columns indices (intyp = 0, 1) or start positions of columns (intyp = 2)
a(1:nz) (R8)	Nonzero elements of the matrix (triplets, row-wise or column-wise)
lb(1:n) (R8)	Lower bound for structural variables
ub(1:n) (R8)	Upper bound for structural variables
rl(1:m) (R8)	Lower bound for constraints
ru(1:m) (R8)	Upper bound for constraints
c(1:n) (R8)	Objective function coefficient
type(1:n) (I4)	Type of structural variable
fcpi(1:n) (R8)	fix charge cost for SC- and SI-variables, threshold values for PI-variables and zero for the rest

Back to index

Optimize

Optimize (Direction, Status, Phase, ObjFunc) optimizes the LP/IP model presently in memory. If the model is an IP model the IP run starts automatically after the model has been solved initially as an LP. If an error occurs the return code gives further information. The output variable *Phase* states from which optimization phase (LP or IP) the function returns.

Input

Direction (I4)	Optimization direction: $\geq$ 0: Minimization < 0: Maximization

Output

Status (I4)	Solution status 0: Optimal LP or IP solution found. 1: LP or IP is infeasible. 2: LP model is unbounded. 3: Only a suboptimal LP/IP solution has been found, when a resource limit (e.g. time limit) was reached. 4: No LP/IP solution has been found, when a resource limit (e.g. time limit) was reached.

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

Phase (I4)	Current phase of optimization 0: LP 1: IP
ObjFunc (R8)	Objective function value of the LP/IP solution (if Status = 0 or 3)
Rc (I4)	0: OK 1: Resource limit (time, memory, iterations, nodes, number of IP solutions) 2: Input error (e.g. error in model) 3: Output error (e.g. no hard disk space) 4: Numerical problem

Example
Running an IP optimization (minimization). The model has already been loaded.

```
rc = Optimize(0, stat, phase, funct)
if rc <> 0 exit
If phase = 0 Then
      ' Stopping in LP phase. This can happen:
      ' 1. If an error occurred when solving the initial LP
      ' 2. If the model is an LP
      ' 3. an IP model is solved as an LP (xlpmip=0)
      If stat = 0 Then
            Output "Optimal LP Solution found. Value=" & funct
            rc = GetLPSolution(stat, funct, Activity(1), RedCost(1),
Status(1))
      Else If stat = 1 Then
            Output "LP is infeasible"
      Else If stat = 2 Then
            Output "LP is unbounded"
      Else
            Output "LP not solved. rc=" & r
      End If
Else
      ' Report results for IP phase (LP was solved to optimality)
      If stat = 0 Then
            Output "Optimal IP Solution found. Value = & funct
            rc = GetIPSolution(stat, funct, Activity(1), RedCost(1),
Status(1))
            Output "index    activity    reduced costs    status"
            'NoCols has been retrieved previously by GetDim
            For j = 1 To NoCols
                  Output j & " " & daActivity(j) & " " & daRedCost(j) & " "
&                       laStatus(j)
            Next
      ElseIf stat = 1 Then
            Output "IP is infeasible"
      ElseIf stat = 3 Then
            Output "IP Solution found - search not completed. Value=" &
funct
      Else
            Output "No IP Solution found - search not completed. rc=" & r
      End If
End If
```

Back to index

PutModel

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

PutModel (Intyp, inf, m, n, nz, ia, ja, a, lb, ub, c, type) transfers a complete LP/IP model to the MOPS.DLL. A model with m constraints and n structural variables must be in the following form:

$$\begin{aligned} \min/\max \quad & c' x \\ bl \;&\leq A\, x \leq bu \\ l \;&\leq \quad x \leq u \end{aligned}$$

with x_j *being integer for every element j from set JI*

The vectors x, c, l, u are real vectors of size *n, A* is a *m x n* matrix of type real, *bl* and *bu* are real *m* -vectors. The vectors *l, u, bl, bu* can have elements, that are plus or minus infinite (+/- inf), to represent unbounded variables or constraints. For equations the values *bl* and *bu* must be identical. *I = {1,..,m}* and *J = {1,..,n}* are index sets for constraints and variables. *JI* is a subset of *J* and contains all indices of integer variables. If *JI* is empty, the model is pure LP. The vector *l* must be stored in positions 1,...,n of the input array lb and *bl* in positions *n+1,..,n+m* of *lb*. Similarly the vector *u* must be stored in positions 1,...,n of the input array *ub* and *bu* in positions *n+1,..,n+m* of *ub*

Input

Intyp (I4)	Mode of storing the model in the returned arrays *ia, ja, a*: 0: Triplets in *ia, ja, a*. The triplets *(i,j,aij)* have no special order. 1: Row-wise storage, where *ia* contains in positions *1,..,m* the start position of the rows in arrays *ja* and *a*. 2: Column-wise storage, where *ja* contains in positions *1,..,n* the start positions of the columns in arrays *ia* und *a*.
inf (R8)	Value that represents infinity (e.g. 1.0E20)
m (I4)	Number of rows
n (I4)	Number of columns
nz (I4)	Number of nonzeros
ia(1:nz) (I4)	Row indices (intyp = 0, 2) or start positions of rows (intyp = 1)
ja(1:nz) (I4)	Columns indices (intyp = 0, 1) or start positions of columns (intyp = 2)
a(1:nz) (R8)	Nonzero elements of the matrix (triplets, row-wise or column-wise)
lb(1:n+m) (R8)	Lower bound for structural variables and constraints
ub(1:n+m) (R8)	Upper bound for structural variables and constraints
c(1:n) (R8)	Objective function coefficient
type(1:n) (I4)	Type of structural variable (0: continuous, <>0 integer)

Output

Rc (I4)	0: OK. 1: The size of model passed is to large for the available memory. 2: Input data error

Example
The following little model will be passed to MOPS.DLL in triplet form (intyp = 0):

Max 3x1 + 5x2
x1 - 20x2 ≤ 0
x1 - 10x2 ≥ 2

x1 ≥ 0, x2 is a binary variable

Parameter for PutModel():

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

```
m=2, n=2, nz=4, inf=1.e20
c = (3.0, 2.0)
typ = (0, 1)
lb = (0.0, 0.0, -1.e20, 2.0)
ub = (1.e20, 1.0, 0.0, 1.e20)
ia = (1, 1, 2, 2)
ja = (1, 2, 1, 2)
a = (1.0, -20.0, 1.0, -10.0)
```

Back to index

ReadMpsFile

ReadMpsFile (FileName) reads an MPS model into the main memory. A path can be optionally specified in *FileName*. Before calling this function you have to call *AllocateMemory() and InitModel().*

Input

FileName (C255)	Name of the MPS file (optionally with path)

Output

Rc (I4)	0: OK. <>0: Error (I/O or syntactical error in MPS file)

Example

```
Rc = ReadMpsFile("c:\mops\models\myfile.mps")
If Rc <> 0 Then
   Rc = GetParameter ("xertyp", Errortype)
   Rc = GetParameter ("xline1", ErrorDescription1)
   Rc = GetParameter ("xline2", ErrorDescription2)
End If
```

Back to index

ReadProfile

ReadProfile (FileName) reads a MOPS Profile. A profile is a plain ASCII file, where you can set any MOPS parameters. See section "Profiles" for an example.

Input

FileName (C255)	Filename of the profile (optionally with path)

Output

Rc (I4)	0: OK <> 0: Error (I/O or syntactical)

Example

```
Rc = ReadProfile("c:\mops\xmops.pro")
If Rc <> 0 Then
   Rc = GetParameter ("xertyp", Errortype)
   Rc = GetParameter ("xline1", ErrorDescription1)
   Rc = GetParameter ("xline2", ErrorDescription2)
End If
```

Back to index

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

ReadTripletFile

ReadTripletFile (FileName) reads triplet file into the main memory. Before calling this function you have to call AllocateMemory() and InitModel(). See section "Triplet Files" for an example.If an error occurs, check parameter *xertyp* for further information.

Input

FileName (C255)	Name of the triplet file (optionally with path)

Output

Rc (I4)	0: OK <> 0: Error (I/O or syntactical) Error types: 1. File I/O error. 2. Max model dimensions xmmax, xnmax, xnzmax, xrcmax exceeded. 3. Syntax error (Bad integer or floating point value, token longer than 64 characters, wrong row index or column index). 4. Logical error (e.g. lb > ub)

Example

```
Rc = ReadTripletFile("d:\mops\models\myfile.tri")
If Rc <> 0 Then
   Rc = GetParameter ("xertyp", Errortype)
   Rc = GetParameter ("xline1", ErrorDescription1)
   Rc = GetParameter ("xline2", ErrorDescription2)
End If
```

Back to index

SetParameter

SetParameter (s) sets MOPS parameters. All parameters are initialized to defaults when calling *InitModel()* and can be set individually afterwards. Several parameters can be specified if they fit into the input string s.

Input

s (C255)	Parameter and value. Syntax: <Parameter>=<Valuet>

Output

Rc (I4)	0: OK <> 0: Error

Example

Set parameter *xoutlv* to 1, so that no log messages are written to the logfile: Set parameter xoutsl = 0 implies that no solution file is written.

```
s = " xoutlv = 1 xoutsl = 0 "
Rc = SetParameter (s)
```

Back to index

WriteMpsFile

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

WriteMpsFile (FileName) saves the model in main memory as MPS file to the hard disk..

Input

FileName (C255)	Name of the MPS file (optionally with path)

Output

Rc (I4)	0: OK <>0: Error 10: Model locked

Example

```
Rc = WriteMpsFile ("c:\mops\models\myfile.mps")
```

Back to index

WriteTripletFile

WriteTripletFile (FileName) saves the current model as a *triplet file* to the hard disk.

Input

FileName (C255)	Name of the triplet file (optionally with path)

Output

Rc (I4)	0: OK <>0: Error 10: Model locked

Example

```
Rc = WriteTripletFile ("c:\mops\models\myfile.tri")
```

Back to index

AllocateMemory

This function is discontinued and only for backward compatibility. Use *Initialize()* instead.

AllocateMemory (Memory) allocates a block of continuous main memory and initializes internal default values. You must call AllocateMemory before you use any other functions to build the model.

If a memory block has already been allocated by a previous call to AllocateMemory, re-invoking the functions frees this memory and allocates a new block, all data in the previous block will be lost.

SetMaxLPDim() and *SetMaxIPDim()* change the default maximal dimensions of a model set by AllocateMemory.

Input

Memory (I4)	Size of the memory block in megabyte [MB]. If Mem = 0, the default value of 300 MB will be used.

Output

Rc (I4)	0: OK <> 0: Not enough memory available

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

Example

```
Rc = AllocateMemory (0)      Allocates 300 MB (default) of main memory .
Rc = AllocateMemory (800)    Allocates 800 MB of main memory.
```

Back to index

InitModel

This function is discontinued and only for backward compatibility. Use *Initialize()* instead.

InitModel () initializes a memory block and sets default values for parameters and internal variables. This function must be called after *AllocateMemory().* For maximal model dimensions default values are used or the dimensions previously set by *SetMaxLPDim()* and *SetMaxIPDim()*, if these functions have been called.

Input

-

Output

Rc (I4)	0: OK <> 0: Error

Example

```
Rc = InitModel()
```

Back to index

FreeMemory

This function is discontinued and only for backward compatibility. Use *Finish()* instead.

FreeMemory () is called to free the memory block allocated by *AllocateMemory().* All model data in main memory is lost if you call this function.

Input

-

Output

Rc (I4)	0: OK <> 0: Error

Example

```
Rc = FreeMemory ()
```

Back to index

OptimizeIP

This function is discontinued and only for backward compatibility. Use *Optimize()* instead.

OptimizeIP (FrNod, Nodes, NIntSol, Errtype, ObjFunc, LpBound) starts an IP optimization run. Before

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

calling *OpitimizeIP()* the model has to be solved successfully as an LP by calling *OptimizeLP()*. The IP optimization will be stopped after having processed *FrNod* nodes or when an optimal solution has been found. You can use a loop to call this solution until the problem will be solved; within the loop you can analyze the results found so far or stop the optimization if appropriate.

Input

FrNod (I4)	Number of nodes until next break.

Output

Nodes (I4)	Current number of nodes in Branch & Bound tree
NIntSol (I4)	number of found IP solutions
Errtype (I4)	Error code for problems during optimization (e.g. resource problems) 0: OK <>0: Error
ObjFunc (R8)	Best objective function value found so far
LpBound (R8)	Best possible objective function value of the IP solution
Rc (I4)	0: Optimization successfully terminated -1: LP is infeasible, unbounded or could not be solved because of a resource limit (memory etc.) -2: No integer variables in model 2: Input error (e.g. in basis or tree file) 3: Output error 4: Numerical problem 6: System error 999: Breakpoint reached (optimization can be continued)

Example
The IP optimization is always stopped after having processed 100 nodes and then continued.

```
xfrnod = 100
Do
      rc = OptimizeIP(xfrnod, xnodes, xnints, xertyp, xzbest, xzubnd)
      If rc = 0 Then
            MsgBox "IP optimization successfully finished."
      Else If rc = 1 Then
            MsgBox "Resource problem (memory, time, iterations)."
      End If
Loop Until rc <> 999
```

Back to index

OptimizeLP

This function is discontinued and only for backward compatibility. Use *Optimize()* instead.

OptimizeLP (FrLog, Iter, NInfeas, LPStatus, ObjFunc, SumInfeas) starts the LP optimizer. This function must also be called when optimizing an IP model to solve the initial LP; *OptimizeIP()* will be called afterwards. When using the Simplex engine (see parameter *xlptyp*), the LP optimization will be stopped after *FrLog* iterations. You can use a loop to break and continue the LP optimization run until the model will be solved.

In case of an error, you can use *GetParameter()* to retrieve *xertyp*, *xline1* and *xline2* to get a more specific error description.

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

Input

FrLog (I4)	Number of simplex iterations until next break.

Output

Iter (I4)	Cumulated number of iterations.
NInfeas (I4)	Number of infeasibilities.
LPStatus (I4)	Optimization status: 0: Optimal 1: Infeasible 2: Unbounded
ObjFunc (R8)	Current objective function value.
SumInfeas (R8)	Sum of infeasibilities.
Rc (I4)	0: Model optimally solved 1: Resource limit reached (memory iterations, time) 2: Input error 3: Output error 4: Numerical problem 6: System error 8: Error while reading or writing IMR to file 999: Breakpoint reached (optimization can be continued)

Example
The optimization of the LP is stopped after each 100 iterations.

```
xfrlog = 100
Do
      rc = OptimizeLP (xfrlog, xiter, xnif, xlpsta, xfunct, xsif)
      If rc = 0 Then
            MsgBox "LP optimization successfully finished."
      Else If rc = 1 Then
            MsgBox "Resource limit (e.g. iteration limit)"
      Else If rc = 2 Then
            MsgBox "Input error occurred."
      Else If rc = 3 Then
            MsgBox "Output error occurred."
      End If
Loop Until rc <> 999
```

Back to index

2.4 Extended functions

2.4.1 Extended functions

Alphabetical list of MOPS.DLL extended functions

Many of the following DLL functions have been added to MOPS.DLL since v8.x.

Index

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

ChangeColLB	ChangeColType	ChangeColUB
ChangeCost	ChangeLhs	ChangeNonzeros
ChangeRhs	DelCol	DelRow
FindName	GenFileNames	GetCol
GetColBase	GetColIPSolution	GetColLB
GetColLPSolution	GetColType	GetColUB
GetCost	GetDim	GetIPObjValue
GetLPObjValue	GetMaxDim	GetNonzero
GetNumberOfColumns	GetNumberOfNZ	GetNumberOfRows
GetRedCost	GetRow	GetRowBase
GetRowDualValues	GetRowIPSolution	GetRowLPSolution
GetRowLhs	GetRowRhs	GetRowType
GetSolutionStatus	PutCol	PutNonzeros
PutRow	SetMaIPDim	SetMaxLPDim

ChangeColLB

ChangeColLB (ColIndex, LB) changes the lower bound of the structural variable with index *ColIndex.*

Input

ColIndex (I4)	Column index
LB (R8)	Lower bound

Output

Rc (I4)	0: OK 1: Bad column index 2: Lower bound is greater than upper bound 10: Model locked

Example
Change lower bound of column 2 to 0:

```
Rc = ChangeColLB(2, 0)
```

Back to index

ChangeColType

ChangeColLB (ColIndex, LB) changes the lower bound of the structural variable with index *ColIndex.*

Input

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

ColIndex (I4)	Column index
LB (R8)	Lower bound

Output

Rc (I4)	0: OK 1: Bad column index 2: Lower bound is greater than upper bound 10: Model locked

Example
Change lower bound of column 2 to 0:

```
Rc = ChangeColLB(2, 0)
```

Back to index

ChangeColUB

ChangeColType (ColIndex, ColType) changes the type of a variable specified by *ColIndex.*

Input

ColIndex (I4)	Column index
ColType (I4)	Type of variable: 0: continuous 1: integer

Output

Rc (I4)	0: OK 1: Bad column index 10: Model locked

Example
Change type of column 3 to integer:

```
Rc = ChangeColType(3, 1)
```

Back to index

ChangeCost

ChangeCost (ColIndex, Cost) changes the objective function coefficient of the structural variable with index *ColIndex.*

Input

ColIndex (I4)	Column index
Cost (R8)	Objective function coefficient

Output

Rc (I4)	0: OK 1: Bad column index 10: Model locked

Example

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

Changes cost of column 3 to 0:

```
Rc = ChangeCost(3, 0)
```

Back to index

ChangeLhs

ChangeLhs (RowIndex, Lhs) changes the lower bound (left hand side) of the constraint with index *RowIndex.*

Input

RowIndex (I4)	Row index
Lhs (R8)	Lower bound (left hand side)

Output

Rc (I4)	0: OK 1: Bad row index 2: Lower bound is greater than upper bound 10: Model locked

Example
Sets the lower bound of row 1 to 0:

```
r = ChangeLhs(1, 0)
```

Back to index

ChangeNonzeros

ChangeNonzeros (NElements, RowIndices, ColIndices, NZs) changes a number of nonzero elements. The last three parameters are arrays that contain the nonzero elements with their row and column indices triplet-wise.

Input

NElements (I4)	Number of elements to be changed
RowIndices (I4)	Array of row indices
ColIndices (I4)	Array of column indices
NZs (R8)	Array of nonzero elements

Output

Rc (I4)	0: OK 1: Bad row index 2: Bad column index 3: One or more nonzero elements is 0 4: Nonzero element to be changed does not exist in the model 5: No rows or columns present, that can be changed 10: Model locked

Example

```
Change 2 nonzero elements (1,1,1) and (1,2,5):

ia(1) = 1: ja(1) = 1: a(1) = 1
```

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

```
ia(2) = 1: ja(2) = 2: a(2) = 5
Rc = ChangeNonzeros(2, ia(1), ja(1), a(1))
```

Back to index

ChangeRhs

ChangeRhs (RowIndex, Rhs) changes the upper bound (right hand side) of the constraint with index *RowIndex.*

Input

RowIndex (I4)	Row index
Rhs (R8)	Upper bound (right hand side)

Output

Rc (I4)	0: OK 1: Bad row index 2: Lower bound is greater than upper bound 10: Model locked

Example
Sets the upper bound of row 1 to 0:

```
r = ChangeRhs(1, 0)
```

Back to index

DelCol

DelCol (ColIndex) deletes a column of the model.

Input

ColIndex (I4)	Column index

Output

Rc (I4)	0: OK 1: Bad column index 6: Matrix is in false internal format 10: Model locked

Example

```
Rc = DelCol (8)
```

Back to index

DelRow

DelRow (RowIndex) deletes a row of the model.

Input

RowIndex (I4)	Row index

Output

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

Rc (I4)	0: OK 1: Bas row index 6: Matrix is in false internal format 10: Model locked

Example

```
Rc = DelRow (8)
```

Back to index

FindName

FindName (Name, Mode, Index) searches for row and column names and returns the corresponding index if available.

Input

Name(C64)	Row name or column name to be searched for.
Mode (I4)	0 if looking for a row name, else column names.

Output

Index (I4)	Index of row or column. Zero if name could not be found.
Rc (I4)	0: OK <> 0: Error 10: Model locked

Example

```
Rc = FindName ("Constraint_Berlin", 0, Index)
```

Back to index

GenFileNames

GenFileNames (FileName) generates names for all MOPS output files (solution files, statistic files etc.) from a template.

Input

FileName (C64)	Name of the file, that serves as a template

Output

Rc (I4)	0: OK 1: Empty or invalid file name

Example

Generates names like for example "demomodel.lps" (LP solution file), "demomodel.sta" (statistic file) etc. :

```
Rc = GenFileNames("demomodel.mps")
```

Back to index

GetCol

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

GetCol (ColIndex, ColName, ColType, NElements, Cost, LB, UB, Status, Activity, RedCost) returns data of a structural variable.

Input

ColIndex (I4)	Column index

Output

ColName (C64)	Name of the variable (max. 64 characters)
ColType (I4)	Type of the Variable (0: continuous, 1: integer)
NElements (I4)	Number of nonzero elements in the column
Cost (R8)	Objective function coefficient
LB (R8)	Lower bound
UB (R8)	Upper bound
Status (I4)	Status of the variable in solution 1: basic 2: lower bound 3: upper bound 4: fixed
Activity (R8)	Activity (solution value) of the variable
RedCost (R8)	Reduced cost of variable in solution
Rc (I4)	0: OK 1: Bad index 10: Model locked

Example

```
Rc = GetCol (17, ColName, ColType, NoElements, Cost, LB, UB, Status,
Activity, RedCost)
```

Back to index

GetColBase

GetColBase (StartColIndex, EndColIndex, Status) gets the variable status after optimization. The array *Status* will be filled and must have sufficient size, at least *EndColIndex-StartColIndex+1.*

Input

StartColIndex (I4)	Start index of columns
EndColIndex (I4)	End index of columns

Output

Status (I4)	Array of variable status in solution 1: basic 2: lower bound 3: upper bound 4: fixed
Rc (I4)	0: OK 1: Start index greater than end index 2: Start index less than 1 3: End index greater than number of columns 10: Model locked

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

Example
Fills array iaColBase with status indicators of columns 3, 4 and 5.

```
Rc = GetColBase(3, 5, iaColBase)
```

Back to index

GetColIPSolution

GetColIPSolution (StartColIndex, EndColIndex, IPSol) gets the IP solution for a number of columns after successful optimization. The array *IPSol* will be filled and must have sufficient size, at least *EndColIndex-StartColIndex+1.*

Input

StartColIndex (I4)	Start index of columns
EndColIndex (I4)	End index of columns

Output

IPSol (R8)	Array of IP solution values
Rc (I4)	0: OK 1: Start index greater than end index 2: Start index less than 1 3: End index greater than number of columns 4: No IP solution present

Example
Fills array *daIPSol* with IP solution values for columns 3, 4 and 5.

```
Rc = GetColIPSolution(3, 5, daIPSol)
```

Back to index

GetColLB

GetColLB (ColIndex, LB) gets the lower bound of a column.

Input

ColIndex (I4)	Column index

Output

LB (R8)	Lower bound
Rc (I4)	0: OK 1: Bad column index 10: Model locked

Example
Get lower bound of column 1:

```
Rc = GetColLB(1, dLB)
```

Back to index

GetColLPSolution

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

GetColLPSolution (StartColIndex, EndColIndex, LPSol) gets the LP solution for a number of columns after successful optimization. The array *LPSol* will be filled and must have sufficient size, at least *EndColIndex-StartColIndex+1.*

Input

StartColIndex (I4)	Start index of columns
EndColIndex (I4)	End index of columns

Output

LPSol (R8)	Array of LP solution values
Rc (I4)	0: OK 1: Start index greater than end index 2: Start index less than 1 3: End index greater than number of columns

Example
Fills array `daLPSol` with IP solution values for columns 3, 4 and 5.

```
Rc = GetColLPSolution(3, 5, daLPSol)
```

Back to index

GetColType

GetColType (ColIndex, ColType) gets the type of a column.

Input

ColIndex (I4)	Column index

Output

ColType (I4)	Type of variable: 0: continuous 1: integer
Rc (I4)	0: OK 1: Bad column index 10: Model locked

Example
Get type of column 1:

```
r = GetColType(1, lColType)
```

Back to index

GetColUB

GetColUB (ColIndex, UB) gets the upper bound of a column.

Input

ColIndex (I4)	Column index

Output

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

LB (R8)	Upper bound
Rc (I4)	0: OK 1: Bad column index 10: Model locked

Example
Get upper bound of column 1:

```
Rc = GetColUB(1, dUB)
```

Back to index

GetCost

GetCost (ColIndex, Cost) gets the objective function coefficient of a variable.

Input

ColIndex (I4)	Column index

Output

Cost (R8)	Objective function coefficient
Rc (I4)	0: OK 1: Bad column index 10: Model locked

Example
Get cost coefficient of column 1:

```
Rr = GetCost(1, dCost)
```

Back to index

GetDim

GetDim (NRows, NCols, NNz) returns current model dimensions.

Input

-

Output

NRows (I4)	Number of rows
NCols (I4)	Number of columns
NNz (I4)	Number of nonzero elements
Rc (I4)	0: OK <> 0: Error

Example

```
Rc = GetDim (NoRows, NoCols, NoNz)
```

Back to index

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

GetIPObjValue

GetIPObjValue () returns the IP objective function value after optimization.

Input

-

Output

GetIPObjValue (R8)	IP objective function value

Example

```
dObj = GetIPObjValue
```

Back to index

GetLPObjValue

GetLPObjValue () returns the LP objective function value after optimization.

Input

-

Output

GetLPObjValue (R8)	LP objective function value

Example

```
dObj = GetLPObjValue
```

Back to index

GetMaxDim

GetMaxDim (MaxRows, MaxCols, MaxNz) returns the maximal model dimensions.
You can change these values with *SetMaxLPDim()*. Eventually you have to change also the size of the memory block with *AllocateMemory()* beforehand.

Input

-

Output

MaxRows (I4)	Maximal number of constraints
MaxCols (I4)	Maximal number of columns
MaxNz (I4)	Maximal number of nonzero elements
Rc (I4)	0: OK <> 0: Error

Example

```
Rc = GetMaxDim (m, n, nz)
```

Back to index

GetNonzero

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

GetNonzero (RowIndex, ColIndex, Element) returns a single nonzero element from the matrix.

Input

RowIndex (I4)	Row index
ColIndex (I4)	Column index

Output

Element (R8)	Matrix element
Rc (I4)	0: OK 1: Bad index 10: Model locked

Example

```
Rc = GetNonzero( 1, 2, Element)
```

reads nonzero in row 1, column 2 and stores the coefficient in *Element*.

Back to index

GetNumberOfColumns

GetNumberOfColumns (NCols) returns the number of columns of the present model.

Input

-

Output

NCols (I4)	Number of columns
Rc (I4)	0: OK 10: Model locked

Example

```
Rc = GetNumberOfColumns(lNCols)
```

Back to index

GetNumberOfNZ

GetNumberOfNZ (NNz) returns the number of nonzeros of the present model.

Input

-

Output

NNz (I4)	Number of nonzeros
Rc (I4)	0: OK 10: Model locked

Example

```
Rc = GetNumberOfNZ(lNNz)
```

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

Back to index

GetNumberOfRows

GetNumberOfRows (NRows) returns the number of rows of the present model.

Input

-

Output

NRows (I4)	Number of rows
Rc (I4)	0: OK 10: Model locked

Example

```
Rc = GetNumberOfRows(lNRows)
```

Back to index

GetRedCost

GetRedCost (StartColIndex, EndColIndex, RedCost) gets the reduced cost of a number of columns after optimization. The array *RedCost* will be filled and must have sufficient size, at least *EndColIndex-StartColIndex+1.*

Input

StartColIndex (I4)	Start index of columns
EndColIndex (I4)	End index of columns

Output

RedCost (R8)	Array of reduced cost values
Rc (I4)	0: OK 1: Start index greater than end index 2: Start index less than 1 3: End index greater than number of columns

Example

Fills array *daRedCost* with reduced cost values for columns 3, 4 and 5.

```
Rc = GetRedCost(3, 5, daRedCost)
```

Back to index

GetRow

GetRow (RowIndex, RowName, RowType, Lhs, Rhs, Status, Activity, RedCost) returns data for one row.
Wurde das Modell nicht optimal gelöst, sind die Werte für *Status, Activity* und *RedCost* nicht definiert.

The variable *RowName* must be long enough to receive the complete name of the constraint. MOPS.DLL is not able to extend an insufficiently dimensioned string variable. Use "*Dim sRowName as string * 64"* in Visual Basic 6.0 for a proper dimensioning.

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

Input

RowIndex (I4)	Row index

Output

RowName (C64)	Row name (max. 64 characters)
RowType (I4)	Type of constraint 1: Unbounded 2: "<=" 3: ">=" 4: "=" 5: Ranged
Lhs (R8)	Lower bound (left hand side)
Rhs (R8)	Upper bound (right hand side)
Status (I4)	Solution status of constraint 1: Basis 2: Lower bound 3: Upper bound 4: Fix
Activity (R8)	Activity of constraint in solution
RedCost (R8)	Reduced cost (dual values) of constraint in solution
Rc (I4)	0: OK 1: Bad index 10: Model locked

Example

```
Rc = GetRow(17, RowName, RowType, Lhs, Rhs, Status, Activity, RedCost)
```

reads values for row 17.

Back to index

GetRowBase

GetRowBase (StartRowIndex, EndRowIndex, Status) gets the solution status of a constraint after optimization. The array *Status* will be filled and must have sufficient size, at least *EndRowIndex-StartRowIndex+1.*

Input

StartRowIndex (I4)	Start index of rows
EndRowIndex (I4)	End index of rows

Output

Status (I4)	Solution status of constraint 1: Basis 2: Lower bound 3: Upper bound 4: Fix
Rc (I4)	0: OK 1: Start index greater than end index 2: Start index less than 1 3: End index greater than number of rows

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

Example
Fills array *laStatus* with reduced cost values for rows 3, 4 and 5.

```
Rc = GetRowBase(3, 5, laStatus)
```

Back to index

GetRowDualValues

GetRowDualValues (StartRowIndex, EndRowIndex, Duals) gets dual values of a number of rows after optimization. The array *Duals* will be filled and must have sufficient size, at least *EndRowIndex-StartRowIndex+1.*

Input

StartRowIndex (I4)	Start index of rows
EndRowIndex (I4)	End index of rows

Output

Duals (R8)	Array of dual values
Rc (I4)	0: OK 1: Start index greater than end index 2: Start index less than 1 3: End index greater than number of rows

Example
Fills array *daDuals* with dual values values for rows 3, 4 and 5.

```
Rc = GetRowDualValues(3, 5, daDuals)
```

Back to index

GetRowIPSolution

GetRowIPSolution (StartRowIndex, EndRowIndex, IPSol) gets IP solution values of a number of rows after optimization. The array *IPSol* will be filled and must have sufficient size, at least *EndRowIndex-StartRowIndex+1.*

Input

StartRowIndex (I4)	Start index of rows
EndRowIndex (I4)	End index of rows

Output

IPSol (R8)	Array of IP solution values
Rc (I4)	0: OK 1: Start index greater than end index 2: Start index less than 1 3: End index greater than number of rows 4: No IP solution present

Example
Fills array *daIPSol* with IP solution values for rows 3, 4 and 5.

```
Rc = GetRowIPSolution(3, 5, daIPSol)
```

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

Back to index

GetRowLPSolution

GetRowLPSolution (StartRowIndex, EndRowIndex, LPSol) gets LP solution values of a number of rows after optimization. The array *LPSol* will be filled and must have sufficient size, at least *EndRowIndex-StartRowIndex+1.*

Input

StartRowIndex (I4)	Start index of rows
EndRowIndex (I4)	End index of rows

Output

LPSol (R8)	Array of LP solution values
Rc (I4)	0: OK 1: Start index greater than end index 2: Start index less than 1 3: End index greater than number of rows

Example
Fills array *daLPSol* with LP solution values for rows 3, 4 and 5.

```
Rc = GetRowLPSolution(3, 5, daLPSol)
```

Back to index

GetRowLhs

GetRowLhs (RowIndex, Lhs) gets left-hand-side (lower bound) of a row.

Input

RowIndex (I4)	Row index

Output

Lhs (R8)	Lower bound (left hand side)
Rc (I4)	0: OK 1: Bad row index 10: Model locked

Example

```
Rc = GetRowLhs(1, dLhs)
```

Back to index

GetRowRhs

GetRowRhs (RowIndex, Rhs) gets right-hand-side (upper bound) of a row.

Input

RowIndex (I4)	Row index

Output

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

Rhs (R8)	Upper bound (right hand side)
Rc (I4)	0: OK 1: Bad row index 10: Model locked

Example

```
Rc = GetRowRhs(1, dRhs)
```

Back to index

GetRowType

GetRowType (RowIndex, RowType) gets type of a row.

Input

RowIndex (I4)	Row index

Output

RowType (I4)	Type of constraint 1: Unbounded 2: "<=" 3: ">=" 4: "=" 5: Ranged
Rc (I4)	0: OK 1: Bad row index 10: Model locked

Example

```
Rc = GetRowType(1, lRowType)
```

Back to index

GetSolutionStatus

GetSolutionStatus() gets the status of an IP or LP solution.

Input

-

Output

GetSolutionStatus (I4)	-1: Status undefined 0: Optimal LP-solution found 1: LP infeasible 2: LP unbounded 3: LP iteration limit xmiter reached 4: LP time limit for reached 5: IP time limit xmxmin for B&B reached 6: Node LP iteration limit xmitip reached 7: Insufficient space xmxnen for LU factors 8: Node limit xmxnod reached 9: Nodes buffer size too small 10: Disk full 11: Granted disk space xmxdsk exhausted

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

	12: xmxnen must be larger than xnzero 13: Input error in IMR or basis 14: Output error 15: Numerical problem 16: frlog limit reached 17: Internal system error 18: Matrix singular 19: B&B finished, optimal solution found 20: B&B finished, no solution found 21: Node-, iteration- or time-limit in b&b reached 22: Input error in b&b tree or IMR 23: Output error in IP optimization 24: Numerical problem in IP optimization 25: Not enough memory for LU-matrices 26: Internal system error in IP optimization

Example

```
lStat = GetSolutionStatus
```

PutCol

PutCol (ColIndex, Mode, ColName, ColType, Cost, LB, UB) adds a new column to the model (when *Mode = 0*) or updates an existing column (when *Mode <> 0*). A new column will be inserted before the passed index ColIndex, i.e. if a new column 17 is inserted, the present column 17 becomes column 18, 18 becomes 19 and so on.

Input

ColIndex (I4)	Column index (values from 1 to number of columns +1)
Mode (I4)	0: Insert 1: Update
ColName	Column name (max. 64 characters)
ColType (I4)	Type of variable: 0: continuous 1: integer
Cost (R8)	Objective function coefficient
LB (R8)	Lower bound
UB (R8)	Upper bound

Output

Rc (I4)	0: OK 1: Bad column index 2: Not enough memory 3: Lower bound is greater than upper bound 4: Not able to check for same column names 5: Another column already has got the same name 10: Model locked Note: Return codes 4 and 5 serve only for information purposes, they are no severe errors and the process of inserting or updating a column will be continued.

Example

```
Rc = PutCol (17, 0, "MyCol_17", 1, 10.0, 0, 100)
```

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

Back to index

PutNonzeros

PutNonzeros (NElements, RowIndices, ColIndices, NZs) adds *NElements* nonzeros to the model. Before using this function you must define rows and columns.

If you want to modify a nonzero coefficient *NElements* must be 1, so only single nonzeros can be modified.
For deleting a nonzero element, call *PutNonzeros()* with *Aij < xdropm (Aji = 0).*

Input

NElements (I4)	Number of elements to be transferred (NElements <= xmaxnz)
RowIndices (I4)	Array of row indices
ColIndices (I4)	Array of column indices
NZs (R8)	Array of nonzero elements

Output

Rc (I4)	0: OK. 1: No constraints defined. 2: No variables defined. 3: Bad row index. 4: Bad column index. 5: Not enough memory. 10: Model locked

Examples
1. Transferring several matrix coefficients:

```
Dim RowIndices() As Long, ColIndices() As Long, Aij() As Double, ...
Redim RowIndices(maxnonzero)
Redim ColIndices (maxnonzero)
Redim Aij (maxnonzero)
Rc = PutNonzeros(NoElements, RowIndices(1), ColIndices(1), Aij(1)).
```

2. Changing a single matrix coefficient:

```
Rc = PutNonzeros(1,4,2,10.0)
```
Replaces element in row 4, column 2 with coefficient 10. If not present the element will be added.

3. Deleting a nonzero element:

```
Rc = PutNonzeros( 1, 4, 2, 0.0)
```

Back to index

PutRow

PutRow (RowIndex, Mode, RowName, Lhs, Rhs) adds a new row to the matrix (if *Mode = 0*) or updates an existing row (if *Mode <> 0*). A new row will be inserted before *RowIndex*, i.e. if a new row 17 is inserted, the present row 17 becomes row 18, 18 becomes 19 and so on.

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

Input

RowIndex (I4)	Row index
Mode (I4)	0: Insert 1: Update
RowName (C64)	Row name (max. 64 characters)
Lhs (R8)	Lower bound (left hand side)
Rhs (R8)	Upper bound (right hand side)

Output

Rc (I4)	0: OK 1: Bad row index 2: Not enough memory 3: Lower bound is greater than upper bound 4: Not able to check for identical row names 5: Another row already has got the same name 10: Model locked Note: Return codes 4 and 5 serve only for information purposes, they are no severe errors and the process of inserting or updating a row will be continued.

Example

1. Adding an unbounded constraint:

```
DIM xinf As Double, lb As Double, ub As Double
Rc = GetParameter("xinf", Value )
xinf = CDbl(Value)
lb = -xinf
ub = xinf
Rc = PutRow(RowIndex, 0, "MyRow_1", lb, ub)
```

2. Adding a "<= 5" constraint.

```
lb = -xinf
ub = 5
Rc = PutRow(lRowIndex, 0, "MyRow_2", lb, ub)
```

Back to index

SetMaxIPDim

SetMaxIPDim (NCli, NImp, NZcl, NNod) changes the maximal IP model dimensions defined in AllocateMemory(). The maximal dimensions of the LP are not changed. If a value is zero, its default value will be used. Call *SetMaxIPDim()* before building the model and calling *Optimize()*.

Input

NCli (I4)	Max. number of cliques
NImp (I4)	Max. number of entries for clique and implications
NZcl (I4)	Max. number of entries for clique elements
NNo (I4)	Max. number of nodes in main memory node table

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

Output

Rc (I4)	0: OK <> 0: Error

Example
Set the number of nodes of the main memory node table to 2000 and keep defaults for the other parameters:

```
Rc = SetMaxIPDim(0,0,0,2000)
```

Back to index

Back to index

SetMaxLPDim

SetMaxLPDim (mx, nx, nzx, lunzx, rcnx, adm, adn, adnz) changes the maximal dimensions of an LP model set in *AllocateMemory()*. If zero is passed for a parameter, its default value remains unchanged.

Input

mx (I4)	Max. number of constraints
nx (I4)	Max.number of structural variables
nzx(I4)	Max. number of nonzero elements in the matrix
unzx(I4)	Max.number of nonzero elements in LU factorization
rcnx (I4)	Max. number of characters for storing row and column names 0: No names will be stored 1: (mx + nx) * xdfnal > 1: Actual number passed
adm (I4)	Max. number of additional constraints (e.g. for cuts)
adn (I4)	Max. number of additional rows
adnz (I4)	Max. number of additional nonzeros (e.g. for cuts)

Output

Rc (I4)	0: OK <> 0: Error

Example
Set number of nonzeros to 600000 and use defaults for all other parameters:

```
Rc = SetMaxLPDim(0,0,600000,0,1,0,0,0)
```

Back to index

2.5 Embedding the DLL

2.5.1 Declarations

You can use MOPS.DLL from a almost any programming language available under Windows, but most languages require that you declare the functions you are about to use. Usually these declaration are done before the actual program, for example in headers or in declaration sections. We used Hungarian Notation for the names of the function parameters, with the following abbreviations indicating the type of the variable expected by MOPS.DLL: d=double, l=long, i=integer, s=string, a=array. So the variable

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

iaMyVariable is an array of integers.

Declarations of all MOPS.DLL functions are available for the most popular programming languages. Simply copy and paste them into your code:

- Visual Basic 6.0
- Visual C/C++ 6.0
- Visual Basic .NET
- C#

See also the demos, that are distributed with MOPS.DLL for examples how to embed MOPS.DLL and how to call functions.

2.5.2 Visual Basic 6.0

Copy and paste these declarations to your VB 6.0 module. Names of functions are case-sensitive, you should not modify them. Eventually the location of the MOPS.DLL must be adapted.

```
Declare Function AllocateMemory Lib "mops.dll" (ByVal lMemory&) As Long
Declare Function DelCol Lib "mops.dll" (ByVal lColIndex&) As Long
Declare Function DelRow Lib "mops.dll" (ByVal lRowIndex&) As Long
Declare Function FindName Lib "mops.dll" (ByVal sName$, ByVal lMode&, lIndex&) As Long
Declare Function Finish Lib "mops.dll" () As Long
Declare Function FreeMemory Lib "mops.dll" () As Long
Declare Function GetCol Lib "mops.dll" (ByVal lColIndex&, ByVal sColName$, lColType&,
                  lNElements&, dCost#, dLB#, dUB#, lStatus&, dActivity#, dRedCost#) As Long
Declare Function GetDim Lib "mops.dll" (lNRows&, lNCols&, lNNz&) As Long
Declare Function GetIPSolution Lib "mops.dll" (lIPSta&, dIPFunct#, daActivity#, daDj#, laSta&)
                  As Long
Declare Function GetLPSolution Lib "mops.dll" (lLPSta&, dLPFunct#, daActivity#, daDj#, laSta&)
                  As Long
Declare Function GetMaxDim Lib "mops.dll" (lMaxRows&, lMaxCols&, lMaxNz&) As Long
Declare Function GetModel Lib "mops.dll" (ByVal lIntype&, INF#, lNRows&, lNCols&, lNNz&,
                  laRowIndices&, laColIndices&, daNZs#, daLBs#, daUBs#, daCosts#, laColTypes&)
                  As Long
Declare Function GetNonzero Lib "mops.dll" (ByVal lRowIndex&, ByVal lColIndex&, dElement#) As
                  Long
Declare Function GetParameter Lib "mops.dll" (ByVal sParameter$, ByVal sValue As String) As Long
Declare Function GetRow Lib "mops.dll" (ByVal lRowIndex&, ByVal sRowName$, lRowType&, dLhs#,
                  dRhs#, lStatus&, dActivity#, dRedCost#) As Long
Declare Function Initialize Lib "mops.dll" () As Long
Declare Function InitModel Lib "mops.dll" () As Long
Declare Function MOPS Lib "mops.dll" (ByVal sFileName$) As Long
Declare Function Optimize Lib "mops.dll" (ByVal lDirection&, lStatus&, lPhase&, dObjFunc#) As
                  Long
Declare Function OptimizeIP Lib "mops.dll" (ByVal lFrNod&, lNodes&, lNIntSol&, lErrtype&,
                  dObjFunc#, dLpBound#) As Long
Declare Function OptimizeLP Lib "mops.dll" (ByVal lFrLog&, lIter&, lNInfeas&, lLPStatus&,
                  dObjFunc#, dSumInfeas#) As Long
Declare Function PutCol Lib "mops.dll" (ByVal lColIndex&, ByVal lMode&, ByVal sColName$, ByVal
                  lColType&, dCost#, dLB#, dUB#) As Long
Declare Function PutModel Lib "mops.dll" (ByVal lIntype&, ByVal INF#, ByVal lNRows&, ByVal
                  lNCols&, ByVal lNNz&, laRowIndices&, laColIndices&, daNZs#, daLBs#, daUB#,
                  daCosts#, laColTypes&) As Long
Declare Function PutNonzeros Lib "mops.dll" (ByVal lNElements&, laRowIndices&, laColIndices&,
                  daNZs#) As Long
Declare Function PutRow Lib "mops.dll" (ByVal lRowIndex&, ByVal lMode&, ByVal sRowName$, dLhs#,
                  dRhs#) As Long
Declare Function ReadMpsFile Lib "mops.dll" (ByVal sFileName$) As Long
Declare Function ReadProfile Lib "mops.dll" (ByVal sFileName$) As Long
```

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

```
Declare Function ReadTripletFile Lib "mops.dll" (ByVal sFileName$) As Long
Declare Function SetMaxIPDim Lib "mops.dll" (ByVal lNCli&, ByVal lNImp&, ByVal lNZcl&, ByVal
                  lNNo&) As Long
Declare Function SetMaxLPDim Lib "mops.dll" (ByVal lNRows&, ByVal lNCols&, ByVal lNNz&, ByVal
                  lNLunz&, ByVal lRcn&, ByVal lNAdRows&, ByVal lNAdCols&, ByVal lNAdNz&) As
                  Long
Declare Function SetParameter Lib "mops.dll" (ByVal sParameter$) As Long
Declare Function WriteMpsFile Lib "mops.dll" (ByVal sFileName$) As Long
Declare Function WriteTripletFile Lib "mops.dll" (ByVal sFileName$) As Long
```

```
Declare Function ChangeColLB Lib "mops.dll" (ByVal lColIndex&, ByVal dLB#) As Long
Declare Function ChangeColType Lib "mops.dll" (ByVal lColIndex&, ByVal lColType&) As Long
Declare Function ChangeColUB Lib "mops.dll" (ByVal lColIndex&, ByVal dUB#) As Long
Declare Function ChangeCost Lib "mops.dll" (ByVal lColIndex&, ByVal dCost#) As Long
Declare Function ChangeLhs Lib "mops.dll" (ByVal lRowIndex&, ByVal dLhs#) As Long
Declare Function ChangeNonzeros Lib "mops.dll" (ByVal lNElements&, laRowIndices&, laColIndices&,
                  daNZs#) As Long
Declare Function ChangeRhs Lib "mops.dll" (ByVal lRowIndex&, ByVal dRhs#) As Long
Declare Function GenFileNames Lib "mops.dll" (ByVal sFileName$) As Long
Declare Function GetColBase Lib "mops.dll" (ByVal lStartColIndex&, ByVal lEndColIndex&,
                  laStatus&) As Long
Declare Function GetColIPSolution Lib "mops.dll" (ByVal lStartColIndex&, ByVal lEndColIndex&,
                  daIPSol#) As Long
Declare Function GetColLB Lib "mops.dll" (ByVal lColIndex&, dLB#) As Long
Declare Function GetColLPSolution Lib "mops.dll" (ByVal lStartColIndex&, ByVal lEndColIndex&,
                  daLPSol#) As Long
Declare Function GetColType Lib "mops.dll" (ByVal lColIndex&, lColType&) As Long
Declare Function GetColUB Lib "mops.dll" (ByVal lColIndex&, dUB#) As Long
Declare Function GetCost Lib "mops.dll" (ByVal lColIndex&, dCost#) As Long
Declare Function GetIPObjValue Lib "mops.dll" () As Double
Declare Function GetLPObjValue Lib "mops.dll" () As Double
Declare Function GetNumberOfColumns Lib "mops.dll" (lNCols&) As Long
Declare Function GetNumberOfNZ Lib "mops.dll" (lNNz&) As Long
Declare Function GetNumberOfRows Lib "mops.dll" (lNRows&) As Long
Declare Function GetRedCost Lib "mops.dll" (ByVal lStartColIndex&, ByVal lEndColIndex&,
                  daRedCost#) As Long
Declare Function GetRowBase Lib "mops.dll" (ByVal lStartRowIndex, ByVal lEndRowIndex, laStatus&)
                  As Long
Declare Function GetRowDualValues Lib "mops.dll" (ByVal lStartRowIndex&, ByVal lEndRowIndex&,
                  daDuals#) As Long
Declare Function GetRowIPSolution Lib "mops.dll" (ByVal lStartRowIndex&, ByVal lEndRowIndex&,
                  daIPSol#) As Long
Declare Function GetRowLPSolution Lib "mops.dll" (ByVal lStartRowIndex&, ByVal lEndRowIndex&,
                  daLPSol#) As Long
Declare Function GetRowLhs Lib "mops.dll" (ByVal lRowIndex&, dLhs#) As Long
Declare Function GetRowRhs Lib "mops.dll" (ByVal lRowIndex&, dRhs#) As Long
Declare Function GetRowType Lib "mops.dll" (ByVal lRowIndex&, lRowType&) As Long
Declare Function GetSolutionStatus Lib "mops.dll" () As Long
```

2.5.3 Visual Basic .NET

Copy and paste these declarations to your VB.NET module. Names of functions are case-sensitive, you should not modify them. Eventually the location of the MOPS.DLL must be adapted. See Samples and VB.NET documentation for usage of IntPtr for passing Arrays to the unmanaged MOPS.DLL.

```
Declare Function AllocateMemory Lib "mops.dll" (ByVal lMemory As Integer) As Integer
Declare Function DelCol Lib "mops.dll" (ByVal lColIndex As Integer) As Integer
Declare Function DelRow Lib "mops.dll" (ByVal lRowIndex As Integer) As Integer
Declare Function FindName Lib "mops.dll" (ByVal sName As String, ByVal lMode As Integer, ByRef
                  lIndex As Integer) As Integer
Declare Function Finish Lib "mops.dll" () As Integer
Declare Function FreeMemory Lib "mops.dll" () As Integer
Declare Function GetCol Lib "mops.dll" (ByVal lColIndex As Integer, ByVal sColName As
                  StringBuilder, ByRef lColTyp As Integer, ByRef lNElements As Integer, ByRef
                  dCost As Double, ByRef dLB As Double, ByRef dUB As Double, ByRef lStatus As
                  Integer, ByRef dActivity As Double, ByRef dRedCost As Double) As Integer
Declare Function GetDim Lib "mops.dll" (ByRef lNRows As Integer, ByRef lNCols As Integer, ByRef
                  lNNz As Integer) As Integer
Declare Function GetIPSolution Lib "mops.dll" (ByRef lIPSta As Integer, ByRef dLPFunct As Double,
                  ByVal pdXs As IntPtr, ByVal pdDj As IntPtr, ByVal lSta As IntPtr) As Integer
```

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

```
Declare Function GetLPSolution Lib "mops.dll" (ByRef lIPSta As Integer, ByRef dLPFunct As Double,
                  ByVal pdXs As IntPtr, ByVal pdDj As IntPtr, ByVal lSta As IntPtr) As Integer
Declare Function GetMaxDim Lib "mops.dll" (ByRef lMaxRows As Integer, ByRef lMaxCols As Integer,
                  ByRef lMaxNz As Integer) As Integer
Declare Function GetModel Lib "mops.dll" (ByVal lIntyp As Integer, ByRef dInf As Double, ByRef
                  lNRows As Integer, ByRef lNCols As Integer, ByRef lNNz As Integer, ByVal
                  plRowIndices As IntPtr, ByVal plColIndices As IntPtr, ByVal pdAij As IntPtr,
                  ByVal pdLBs As IntPtr, ByVal pdUBs As IntPtr, ByVal pdCosts As IntPtr, ByVal
                  plColTyps As IntPtr) As Integer
Declare Function GetNonzero Lib "mops.dll" (ByVal iRowIndex As Integer, ByVal lColIndex As
                  Integer, ByRef dElement As Double) As Integer
Declare Function GetParameter Lib "mops.dll" (ByVal sParameter As String, ByVal sWert As
                  StringBuilder) As Integer
Declare Function GetRow Lib "mops.dll" (ByVal lRowIndex As Integer, ByVal sRowName As
                  StringBuilder, ByRef lRowTyp As Integer, ByRef dLoRhs As Double, ByRef dUpRhs
                  As Double, ByRef lStatus As Integer, ByRef dActivity As Double, ByRef dRedCost
                  As Double) As Integer
Declare Function Initialize Lib "mops.dll" () As Integer
Declare Function InitModel Lib "mops.dll" () As Integer
Declare Function Mops Lib "mops.dll" (ByVal fnpro As String) As Integer
Declare Function Optimize Lib "mops.dll" (ByVal lDirection As Integer, ByRef lStatus As Integer,
                  ByRef lPhase As Integer, ByRef dZf As Double) As Integer
Declare Function OptimizeIP Lib "mops.dll" (ByVal lFrNod As Integer, ByRef lNodes As Integer,
                  ByRef lNIntSol As Integer, ByRef lResou As Integer, ByRef dZf As Double, ByRef
                  dLpBound As Double) As Integer
Declare Function OptimizeLP Lib "mops.dll" (ByVal lFrLog As Integer, ByRef lIter As Integer,
                  ByRef lNoInfeas As Integer, ByRef lLPStatus As Integer, ByRef dZf As Double,
                  ByRef dSumInfeas As Double) As Integer
Declare Function PutCol Lib "mops.dll" (ByVal lColIndex As Integer, ByVal lMode As Integer, ByVal
                  sColName As String, ByVal lColTyp As Integer, ByRef dCost As Double, ByRef dLB
                  As Double, ByRef dUB As Double) As Integer
Declare Function PutModel Lib "mops.dll" (ByVal lIntyp As Integer, ByVal dInf As Double, ByVal
                  lNRows As Integer, ByVal lNCols As Integer, ByVal lNNz As Integer, ByVal
                  plRowIndices As IntPtr, ByVal plColIndices As IntPtr, ByVal pdAij As IntPtr,
                  ByVal pdLBs As IntPtr, ByVal pdUBs As IntPtr, ByVal pdCosts As IntPtr, ByVal
                  plColTyps As IntPtr) As Integer
Declare Function PutNonzeros Lib "mops.dll" (ByVal lNElements As Integer, ByVal plRowIndices As
                  IntPtr, ByVal plColIndices As IntPtr, ByVal pdAij As IntPtr) As Integer
Declare Function PutRow Lib "mops.dll" (ByVal iRowIndex As Integer, ByVal lMode As Integer, ByVal
                  sRowName As String, ByRef dLoRhs As Double, ByRef dUpRhs As Double) As Integer
Declare Function ReadMpsFile Lib "mops.dll" (ByVal sFileName As String) As Integer
Declare Function ReadProfile Lib "mops.dll" (ByVal sFileName As String) As Integer
Declare Function SetMaxIPDim Lib "mops.dll" (ByVal lMMax As Integer, ByVal lNMax As Integer,
                  ByVal lNZMax As Integer) As Integer
Declare Function SetMaxLPDim Lib "mops.dll" (ByVal lMMax As Integer, ByVal lNMax As Integer,
                  ByVal lNZMax As Integer) As Integer
Declare Function SetParameter Lib "mops.dll" (ByVal s As String) As Integer
Declare Function WriteMpsFile Lib "mops.dll" (ByVal sFileName As String) As Integer
Declare Function WriteTripletFile Lib "mops.dll" (ByVal sFileName As String) As Integer
```

```
Declare Function GetIPObjValue Lib "mops.dll" () As Double
Declare Function GetLPObjValue Lib "mops.dll" () As Double
Declare Function ChangeColLB Lib "mops.dll" (ByVal lColIndex As Integer, ByVal dLB As Double) As
                  Integer
Declare Function ChangeColType Lib "mops.dll" (ByVal lColIndex As Integer, ByVal lColTyp As
                  Integer) As Integer
Declare Function ChangeColUB Lib "mops.dll" (ByVal lColIndex As Integer, ByVal dUB As Double) As
                  Integer
Declare Function ChangeCost Lib "mops.dll" (ByVal lColIndex As Integer, ByVal dCost As Double) As
                  Integer
Declare Function ChangeNonzeros Lib "mops.dll" (ByVal lNElements As Integer, ByVal plRowIndices
                  As IntPtr, ByVal plColIndices As IntPtr, ByVal pdAij As IntPtr) As Integer
Declare Function ChangeLhs Lib "mops.dll" (ByVal lRowIndex As Integer, ByVal dLhs As Double) As
                  Integer
Declare Function ChangeRhs Lib "mops.dll" (ByVal lRowIndex As Integer, ByVal dRhs As Double) As
                  Integer
Declare Function GenFileNames Lib "mops.dll" (ByVal sFileName As String) As Integer
Declare Function GetColBase Lib "mops.dll" (ByVal lStartColIndex As Integer, ByVal lEndColIndex
                  As Integer, ByVal plSta As IntPtr) As Integer
Declare Function GetColIPSolution Lib "mops.dll" (ByVal lStartIndex As Integer, ByVal lEndIndex
                  As Integer, ByVal pdVal As IntPtr) As Integer
Declare Function GetColLB Lib "mops.dll" (ByVal lColIndex As Integer, ByRef dLB As Double) As
                  Integer
```

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

```
Declare Function GetColLPSolution Lib "mops.dll" (ByVal lStartIndex As Integer, ByVal lEndIndex
                  As Integer, ByVal pdVal As IntPtr) As Integer
Declare Function GetColType Lib "mops.dll" (ByVal lColIndex As Integer, ByRef lColTyp As Integer)
                  As Integer
Declare Function GetColUB Lib "mops.dll" (ByVal lColIndex As Integer, ByRef dUB As Double) As
                  Integer
Declare Function GetCost Lib "mops.dll" (ByVal lColIndex As Integer, ByRef dCost As Double) As
                  Integer
Declare Function GetNumberOfColumns Lib "mops.dll" (ByRef lNCols As Integer) As Integer
Declare Function GetNumberOfNZ Lib "mops.dll" (ByRef lNNz As Integer) As Integer
Declare Function GetNumberOfRows Lib "mops.dll" (ByRef lNRows As Integer) As Integer
Declare Function GetRedCost Lib "mops.dll" (ByVal lStartIndex As Integer, ByVal lColIndex As
                  Integer, ByVal pdRedCost As IntPtr) As Integer
Declare Function GetRowBase Lib "mops.dll" (ByVal lStartIndex As Integer, ByVal lEndIndex As
                  Integer, ByVal plSta As IntPtr) As Integer
Declare Function GetRowDualValues Lib "mops.dll" (ByVal lStartIndex As Integer, ByVal lEndIndex
                  As Integer, ByVal pdVal As IntPtr) As Integer
Declare Function GetRowIPSolution Lib "mops.dll" (ByVal lStartIndex As Integer, ByVal lEndIndex
                  As Integer, ByVal pdVal As IntPtr) As Integer
Declare Function GetRowLhs Lib "mops.dll" (ByVal lRowIndex As Integer, ByRef dLhs As Double) As
                  Integer
Declare Function GetRowLPSolution Lib "mops.dll" (ByVal lStartIndex As Integer, ByVal lEndIndex
                  As Integer, ByVal pdVal As IntPtr) As Integer
Declare Function GetRowRhs Lib "mops.dll" (ByVal lRowIndex As Integer, ByRef dRhs As Double) As
                  Integer
Declare Function GetRowType Lib "mops.dll" (ByVal lRowIndex As Integer, ByRef lRowTyp As Integer)
                  As Integer
Declare Function GetSolutionStatus Lib "mops.dll" () As Integer
```

2.5.4 Visual C/C++ 6.0

Copy and paste these declarations into a C/C++ header file, name it for example "mopsdll.h", and include this header into your C/C++ code.

```
//-------------------------------------------------------------------------------------
// mopsdll.h                                                                  PI 01/08
//
// This header has to be included, when the MOPS.DLL is used from C/C++ programs
//-------------------------------------------------------------------------------------

typedef long (__stdcall *ALLOCATEMEMORY)(long);
typedef long (__stdcall *DELCOL)(long);
typedef long (__stdcall *DELROW)(long);
typedef long (__stdcall *FINDNAME)(char*, long, long*);
typedef long (__stdcall *FINISH)(void);
typedef long (__stdcall *FREEMEMORY)(void);
typedef long (__stdcall *GETCOL)(long, char*, long*, long*, double*, double*, double*, long*,
double*, double*);
typedef long (__stdcall *GETDIM)(long*, long*, long*);
typedef long (__stdcall *GETIPSOLUTION)(long*, double*, double*, double*, long*);
typedef long (__stdcall *GETLPSOLUTION)(long*, double*, double*, double*, long*);
typedef long (__stdcall *GETMAXDIM)(long*, long*, long*);
typedef long (__stdcall *GETMODEL)(long, double*, long*, long*, long*, long*, long*, double*,
double*, double*, double*, long*);
typedef long (__stdcall *GETNONZERO)(long, long, double*);
typedef long (__stdcall *GETPARAMETER)(char*, char*);
typedef long (__stdcall *GETROW)(long, char*, long*, double*, double*, long*, double*, double*);
typedef long (__stdcall *INITIALIZE)(void);
typedef long (__stdcall *INITMODEL)(void);
typedef long (__stdcall *MOPS)(char*);
typedef long (__stdcall *OPTIMIZEIP)(long, long*, long*, long*, double*, double*);
typedef long (__stdcall *OPTIMIZELP)(long, long*, long*, long*, double*, double*);
typedef long (__stdcall *OPTIMIZE)  (long, long*, long*, double*);
typedef long (__stdcall *PUTCOL)(long, long, char*, long, double*, double*, double*);
typedef long (__stdcall *PUTMODEL)(long, double, long, long, long, long*, long*, double*,
double*, double*, double*, long*);
typedef long (__stdcall *PUTNONZEROS)(long, long*, long*, double*);
typedef long (__stdcall *PUTOBJ)(long, double);
typedef long (__stdcall *PUTROW)(long, long, char*, double*, double*);
typedef long (__stdcall *READMPSFILE)(char*);
```

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

```
typedef long (__stdcall *READPROFILE)(char*);
typedef long (__stdcall *READTRIPLETFILE)(char*);
typedef long (__stdcall *SETMAXIPDIM)(long, long, long, long);
typedef long (__stdcall *SETMAXLPDIM)(long, long, long, long, long, long, long, long);
typedef long (__stdcall *SETPARAMETER)(char*);
typedef long (__stdcall *WRITEMPSFILE)(char*);
typedef long (__stdcall *WRITETRIPLETFILE)(char*);
```

```
typedef double (__stdcall *GETIPOBJVALUE)();
typedef double (__stdcall *GETLPOBJVALUE)();
typedef long (__stdcall *CHANGECOLLB)(long, double);
typedef long (__stdcall *CHANGECOLTYPE)(long, long);
typedef long (__stdcall *CHANGECOLUB)(long, double);
typedef long (__stdcall *CHANGECOST)(long, double);
typedef long (__stdcall *CHANGENONZEROS)(long, long*, long*, double*);
typedef long (__stdcall *CHANGELHS)(int, double);
typedef long (__stdcall *CHANGERHS)(int, double);
typedef long (__stdcall *GENFILENAMES)(char*);
typedef long (__stdcall *GETCOLBASE)(long, long, long*);
typedef long (__stdcall *GETCOLIPSOLUTION)(long, long, double*);
typedef long (__stdcall *GETCOLLB)(long, double*);
typedef long (__stdcall *GETCOLLPSOLUTION)(long, long, double*);
typedef long (__stdcall *GETCOLTYPE)(long, long*);
typedef long (__stdcall *GETCOLUB)(long, double*);
typedef long (__stdcall *GETCOST)(long, double*);
typedef long (__stdcall *GETNUMBEROFCOLUMNS)(long*);
typedef long (__stdcall *GETNUMBEROFNZ)(long*);
typedef long (__stdcall *GETNUMBEROFROWS)(long*);
typedef long (__stdcall *GETREDCOST)(long, long, double*);
typedef long (__stdcall *GETROWBASE)(long, long, long*);
typedef long (__stdcall *GETROWDUALVALUES)(long, long, double*);
typedef long (__stdcall *GETROWIPSOLUTION)(long, long, double*);
typedef long (__stdcall *GETROWLHS)(long, double*);
typedef long (__stdcall *GETROWLPSOLUTION)(long, long, double*);
typedef long (__stdcall *GETROWRHS)(long, double*);
typedef long (__stdcall *GETROWTYPE)(long, long*);
typedef long (__stdcall *GETSOLUTIONSTATUS)();
```

2.5.5 C#

In C#, MOPS.DLL functions calls must be encapsulated into a class. You call the static methods of this class from the model generator. Add a new class to your project and copy this code to the class file CMops.cs.

```
using System;
using System.Runtime.InteropServices;
using System.Text;

namespace CS_Caller_MOPSDLL
{
        //-------------------------------------------------------------------------
        // C# Inclusion class for MOPS.DLL v9.x                                PI 12/07
        // Copy this class to your C# code and call the static methods of the class
        //
        // Prefixes of variable names indicate type (hungarian notation):
        // i = integer, d = double, s = string, a = array
        //-------------------------------------------------------------------------
 public class CMops
 {
   private const string mopsFile = "mops.dll";

   // All external, unmanaged DLL functions must be static
   [DllImport(mopsFile)] public static extern int AllocateMemory(int iMemory);
   [DllImport(mopsFile)] public static extern int DelCol(int iColIndex);
   [DllImport(mopsFile)] public static extern int DelRow(int iRowIndex);
   [DllImport(mopsFile)] public static extern int FindName(string sName, int iMode,
      ref int iIndex);
   [DllImport(mopsFile)] public static extern int Finish();
   [DllImport(mopsFile)] public static extern int FreeMemory();
   [DllImport(mopsFile)] public static extern int GetCol(int iColIndex,
      StringBuilder sColName, ref int iColType, ref int iNElements, ref double dCost,
```

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

```
    ref double dLB, ref double dUB, ref int iStatus, ref double dActivity,
    ref double dRedCost);
[DllImport(mopsFile)] public static extern int GetDim(ref int iNRows,
    ref int iNCols, ref int iNNz);
[DllImport(mopsFile)] public static extern int GetIPSolution(ref int iIpSta,
    ref double dIpFunct, IntPtr daXs, IntPtr daDj, IntPtr iaSta);
[DllImport(mopsFile)] public static extern int GetLPSolution(ref int iLpSta,
    ref double dLpFunct, IntPtr daXs, IntPtr daDj, IntPtr iaSta);
[DllImport(mopsFile)] public static extern int GetMaxDim(ref int iMaxRows,
    ref int iMaxCols, ref int iMaxNz);
[DllImport(mopsFile)] public static extern int GenFileNames(string sFileName);
[DllImport(mopsFile)] public static extern int GetModel(int iIntype,
    ref double dINF, ref int iNRows, ref int iNCols, ref int iNNz,
    IntPtr iaRowIndices, IntPtr iaColIndices, IntPtr daNZs,
    IntPtr daLBs, IntPtr daUBs, IntPtr daCosts, IntPtr iaColTypes);
[DllImport(mopsFile)] public static extern int GetNonzero(int iRowIndex,
    int iColIndex, ref double dElement);
[DllImport(mopsFile)] public static extern int GetParameter(string sParameter,
    StringBuilder sValue);
[DllImport(mopsFile)] public static extern int GetRow(int iRowIndex,
    StringBuilder sRowName, ref int iRowType, ref double dLhs,
    ref double dRhs, ref int iStatus, ref double dActivity, ref double dRedCost);

[DllImport(mopsFile)] public static extern int Initialize();
[DllImport(mopsFile)] public static extern int InitModel();
[DllImport(mopsFile)] public static extern int MOPS(string sFileName);
[DllImport(mopsFile)] public static extern int Optimize(int iDirection,
    ref int iStatus, ref int iPhase, ref double dObjFunc);
[DllImport(mopsFile)] public static extern int OptimizeIP(int iFrNod,
    ref int iNodes, ref int iNIntSol, ref int iErrtype, ref double dObjFunc,
    ref double dLpBound);
[DllImport(mopsFile)] public static extern int OptimizeLP(int iFrLog,
    ref int iIter, ref int iNInfeas, ref int iLpStatus, ref double dObjFunc,
    ref double dSumInfeas);
[DllImport(mopsFile)] public static extern int PutCol(int iColIndex, int iMode,
    string sColName, int iColType, ref double dCost, ref double dLB,
    ref double dUB);
[DllImport(mopsFile)] public static extern int PutModel(int iIntype, double dINF,
    int iNRows, int iNCols, int iNNz, IntPtr iaRowIndices, IntPtr iaColIndices,
    IntPtr daNZs, IntPtr daLBs, IntPtr daUBs, IntPtr daCosts, IntPtr iaColTypes);
[DllImport(mopsFile)] public static extern int PutNonzeros(int iNElements,
    IntPtr iaRowIndices, IntPtr iaColIndices, IntPtr daNZs);
[DllImport(mopsFile)] public static extern int PutRow(int iRowIndex, int iMode,
    string sRowName, ref double dLhs, ref double dRhs);
[DllImport(mopsFile)] public static extern int ReadMpsFile(string sFileName);
[DllImport(mopsFile)] public static extern int ReadProfile(string sFileName);
[DllImport(mopsFile)] public static extern int ReadTripletFile(string sFileName);
[DllImport(mopsFile)] public static extern int SetMaxIPDim(int iNCli, int iNImp,
    int iNZcl, int iNNo);
[DllImport(mopsFile)] public static extern int SetMaxLPDim(int iNRows, int iNCols,
    int iNNz, int iNLunz, int iRcn, int iNAdRows, int iNAdCols, int iNAdNz);
[DllImport(mopsFile)] public static extern int SetParameter(string sParameter);
[DllImport(mopsFile)] public static extern int WriteMpsFile(string sFileName);
[DllImport(mopsFile)] public static extern int WriteTripletFile(string sFileName);
[DllImport(mopsFile)] public static extern int
    ChangeColLB(int iColIndex, double dLB);
[DllImport(mopsFile)] public static extern int
    ChangeColType(int iColIndex, int iColType);
[DllImport(mopsFile)] public static extern int
    ChangeColUB(int iColIndex, double dUB);
[DllImport(mopsFile)] public static extern int
    ChangeCost(int iColIndex, double dCost);
[DllImport(mopsFile)] public static extern int
    ChangeLhs(int iRowIndex, double dLhs);
[DllImport(mopsFile)] public static extern int
    ChangeNonzeros(int iNElements, IntPtr iaRowIndices, IntPtr iaColIndices,
    IntPtr daNZs);
[DllImport(mopsFile)] public static extern int
    ChangeRhs(int iRowIndex, double dRhs);
[DllImport(mopsFile)] public static extern int
    GetColBase(int iStartColIndex, int iEndColIndex, IntPtr iaStatus);
[DllImport(mopsFile)] public static extern int
    GetColIPSolution(int iStartColIndex, int iEndColIndex, IntPtr daIPSol);
[DllImport(mopsFile)] public static extern int
    GetColLB(int iColIndex, ref double dLB);
[DllImport(mopsFile)] public static extern int
```

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

```
      GetColLPSolution(int iStartColIndex, int iEndColIndex, IntPtr daLPSol);
  [DllImport(mopsFile)] public static extern int
     GetColType(int iColIndex, ref int iColType);
   [DllImport(mopsFile)] public static extern int
      GetColUB(int iColIndex, ref double dUB);
   [DllImport(mopsFile)] public static extern int
      GetCost(int iColIndex, ref double dCost);
   [DllImport(mopsFile)] public static extern double
      GetIPObjValue();
   [DllImport(mopsFile)] public static extern double
      GetLPObjValue();
   [DllImport(mopsFile)] public static extern int
      GetNumberOfColumns(ref int iNCols);
   [DllImport(mopsFile)] public static extern int
      GetNumberOfNZ(ref int iNNz);
   [DllImport(mopsFile)] public static extern int
      GetNumberOfRows(ref int iNRows);
   [DllImport(mopsFile)] public static extern int
      GetRedCost(int iStartColIndex, int iEndColIndex, IntPtr daRedCost);
   [DllImport(mopsFile)] public static extern int
      GetRowBase(int iStartRowIndex, int iEndRowIndex, IntPtr iaStatus);
   [DllImport(mopsFile)] public static extern int
      GetRowDualValues(int iStartRowIndex, int iEndRowIndex, IntPtr daDuals);
   [DllImport(mopsFile)] public static extern int
      GetRowIPSolution(int iStartRowIndex, int iEndRowIndex, IntPtr daIPSol);
   [DllImport(mopsFile)] public static extern int
      GetRowLPSolution(int iStartRowIndex, int iEndRowIndex, IntPtr daLPSol);
   [DllImport(mopsFile)] public static extern int
      GetRowLhs(int iRowIndex, ref double dLhs);
   [DllImport(mopsFile)] public static extern int
      GetRowRhs(int iRowIndex, ref double dRhs);
   [DllImport(mopsFile)] public static extern int
      GetRowType(int iRowIndex, ref int iRowType);
   [DllImport(mopsFile)] public static extern int
      GetSolutionStatus();
   }
}
```

2.6 64 Bit Version

2.6.1 64 Bit Version

The 64-bit version of MOPS.DLL (MOPS64.DLL) works exactly the same way the previously described 32-bit version does. In a 32-bit operating system addresses are 4 bytes long, so when a DLL function expects an address pointer, like for example in PutModel(), a 4 byte pointer is passed. In 64 bit operating systems addresses double to 8 bytes. Since the compiler takes care of these operations the user of 64-bit MOPS.DLL doesn't have to bother about this. So the syntax for calling 64 bit functions is exactly the same as for calling 32-bit functions.

You cannot use the 32-bit MOPS.DLL in a 64-bit process and vice versa. Windows XP x64 has a 32 bit compatibility mode (WOW64), in which 32-bit applications run as if they were running on a native 32 bit machine. Via WOW64 a 32-bit MOPS.DLL can be used under Windows XP x64.

Since the 32 bit MOPS.DLL is limited to 2 Gigabytes of main memory, which id independent of the amount of physical main memory, there is a limit on the size of the models which can be solved, since the model data and a large amount of working data has to be kept in main memory.

The key advantage of using 64 bit-software is the extremely large address space which allows to solve much larger models. Another advantage is that memory intensive algorithms such as interior point methods can be used, where the storage of the Cholesky factorization alone can take several Gigabytes. (see also the MOPS White Paper).

For those models which can be solved with the 32-bit-Versions of MOPS there is no speed advantage compared to the 64 bit versions of MOPS.

The samples include 64-bit projects for VB.NET, C# and C++.

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

Chapter III

MOPS.LIB

3 MOPS.LIB

3.1 Using MOPS.LIB

In addition to the DLL version, MOPS is also available as a static library, the MOPS.LIB. If the question arises which type of MOPS library you should use for your application, the LIB or the DLL, in most cases the answer will be to use the MOPS.DLL. Nevertheless there are cases when utilizing the static library can be appropriate:

- MOPS.LIB resp, MOPS64.LIB offers you full access to all MOPS routines. When using a DLL you can only call functions, that are explicitly exported by the dynamic link library; any other routine is hidden in the DLL. The static library on the other hand allows you to call these "hidden" routines. The Fortran MOPS.LIB demo sample for instance calls *xcordr* to reorder the IMR, a function that would not be accessible in the DLL. Using such MOPS internal routines of course requires that you have expert knowledge of the MOPS architecture, since internal functions are not publicly documented.
- MOPS.LIB allows furthermore full access to all MOPS-internal data structures. You can manipulate all common block variables (even those not supported by *SetParameter()*) and all global arrays. This enables you to write very efficient - though in most cases very intricate - model generators, which build optimization models directly on the IMR arrays. See the MOPS White Paper for a description of the IMR. Again, working with internal data structures requires profound knowledge of MOPS.
- Using LIB functions bypasses the DLL Layer. Internally all MOPS.DLL functions "go down" to call LIB functions after performing DLL specific checks and transformations. Thus using MOPS.LIB is more direct since the DLL Layer overload is stripped. In large model generators there might be a very, very small performance gain from this, though in most cases you would notice no significant performance difference in using the DLL or the LIB.
- The MOPS.LIB resp. MOPS64.LIB must not be explicitly loaded during run time. But since the MOPS.DLL is a very small footprint library of about 2 MB, and since loading is done quickly and efficiently by Windows, there is no measurable performance gain from not having to load anything when using the LIB.
- When linking statically, the used functions from the MOPS.LIB become part of the application executable. This has advantages for application deployment, since you do not need to distribute an additional DLL. Furthermore it becomes impossible to rip the MOPS.DLL from you application and use it independently.

Summarizing, it makes only sense to use the MOPS.LIB if you need full access to MOPS internal functions and data structures. In the vast majority of cases, the DLL is the more appropriate way to employ MOPS in your application.

See section MOPS.LIB Functions for a list of the most important functions available in the static library.

3.2 MOPS.LIB Functions

The functions of the static MOPS library can be called with no further ado from any Fortran program under Windows. Library versions for *HP Visual Fortran (formerly Compaq / Digital)* and for *Intel Fortran* are available.

The MOPS.LIB functions and their calling sequences have been re-designed in MOPS v8.x to make the usage of DLL and LIB more alike. Most MOPS.DLL functions have a MOPS.LIB function as counterpart, which is called with exactly the same parameters. Refer to the description of the MOPS.DLL function if you need information about the analog static library function.

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

Some DLL functions have no counterparts, because in IMR context when using the LIB you have direct access to all MOPS variables, and information like for example model dimensions can be retrieved directly from the variables (in that case *xm, xn, xnzero*). One important difference between DLL and LIB functions is that when passing a string, the LIB functions require that as additional parameter the length of the string is passed just behind the string itself. So DLL function *FindName()* is called *FindName(name, mode, index)* and the corresponding LIB function *xfname* is called *xfname (name, length, mode, index).*

The following table shows MOPS.DLL functions and the corresponding LIB functions. See also the MOPS.LIB demonstrator for samples how to call most LIB functions.

MOPS.DLL functions and their MOPS.LIB counterparts

AllocateMemory	xallocm
DelCol	xdecol
DelRow	xderow
FindName	xfname
FreeMemory	xfreem
GetCol	xgtcol
GetDim	-
GetIPSolution	xgtips
GetLPSolution	xgtlps
GetMaxDim	-
GetModel	xgtmod
GetNonzero	xgtnz
GetParameter	-
GetRow	xgtrow
Initialize	xtrdat, xdefau
InitModel	xinitm
LoadModel	
Mops	-
Optimize	xoptim
OptimizeIP	xoptip
OptimizeLP	xoptlp

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

PutCol	xptcol
PutModel	xptmod
PutNonzeros	xptnzs
PutRow	xptrow
ReadMpsFile	xrdmps
ReadProfile	xrprof
ReadTripletFile	xrdtri
SetMaxIPDim	-
SetMaxLPDim	-
SetParameter	-
WriteMpsFile	xstmps
WriteTripletFile	xsttri

ChangeColLB	xchclb
ChangeColType	xchctype
ChangeColUB	xchcub
ChangeCost	xchcost
ChangeNonzeros	xchnzs
ChangeLhs	xchlhs
ChangeRhs	xchrhs
GenFileNames	xgenfn
GetColBase	xgtcbs
GetColIPSolution	xgtxx
GetColLB	xgtclb
GetColLPSolution	xgtxx
GetColUB	xgtcub
GetColType	xgtctype
GetCost	xgtcost
GetIPObjValue	-
GetLPObjValue	-

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

GetNumberOfColumns	-
GetNumberOfNZ	-
GetNumberOfRows	-
GetRedCost	xgtrco
GetRowBase	xgtrowbs
GetRowDualValues	xgtrco
GetRowLhs	xgtrowlhs
GetRowIPSolution	xgtxx
GetRowLPSolution	xgtxx
GetRowRhs	xgtrowrhs
GetRowType	xgtrowtype
GetSolutionStatus	-

Note

For accessing MOPS.LIB functions in HP/Compaq Visual Fortran 6 environments (former Digital Visual Fortran) you must change two options: *Fortran -> External Procedures -> Argument Passing conventions* must be set to *C, By Reference* and *Runtime Error Checking* for *Array and String Bounds* must be disabled. For Intel applies the same accordingly.

3.3 Calling MOPS.LIB from C/C++

The MOPS.LIB can also be linked against C/C++ programs, but peculiarities of the C/Fortran inter-language interface make this approach somewhat more intricate. The Fortran Common Block must be mapped to an exactly corresponding C structure, pointers to the internal MOPS arrays must be obtained and different calling sequences and name decorations must be considered.

In general the dynamic link library is the easiest and most stable way of using MOPS from a C/C++ application. You should consider a C/Fortran inter-language architecture with MOPS.LIB only if there are very good reasons for it.

If necessary, contact the MOPS development team for more information about this subject.

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

Chapter IV

MOPS.EXE

4 MOPS.EXE

4.1 Using MOPS.EXE

MOPS is also available as a command line executable for Windows. The advantage of the command line executable MOPS.EXE is, that MOPS can be run without any programming effort, independent of any development environment and with no previous installation process. However, the model has to be passed to MOPS as an mps or triplet file. The LP and IP-solution are also stored as files on disk.

MOPS.EXE carries out a complete LP/IP optimization run as specified in a MOPS Profile. To start MOPS from the command line, use the following Syntax, assuming that MOPS.EXE is located in the current directory:

c:\> *mops [myprofile]*

If no profile name is specified, the default name *xmops.pro* will be taken.

In the profile all mops parameters can be set and the model will be read from an MPS or Triplet file, depending on the values specified in *xinfor* and *xfnmps*. Profiles can be edited with any text editor. After optimization, statistics and solutions can be loaded into your favorite editor. See chapter "MOPS Files" for further details.

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

Chapter V

MOPS STUDIO

5 MOPS STUDIO

5.1 Using MOPS STUDIO

MOPS Studio

MOPS Studio is an interactive front end to MOPS which supports the creation of models in the modeling languages AMPL, MathProg, Lindo and MPL combined with MOPS. The version of AMPL, implemented in MOPS Studio, is identical to the original version with an additional graphic user surface. Furthermore models in standard MPS-format or in the MOPS-specific triplet format can be loaded and solved with MOPS.

Index

1. Main Window

2. Menus

3. Toolbar

4. Input Window

5. Output Window

6. Options

7. Parameters

8. Execute AMPL

9. Solution

Main Window

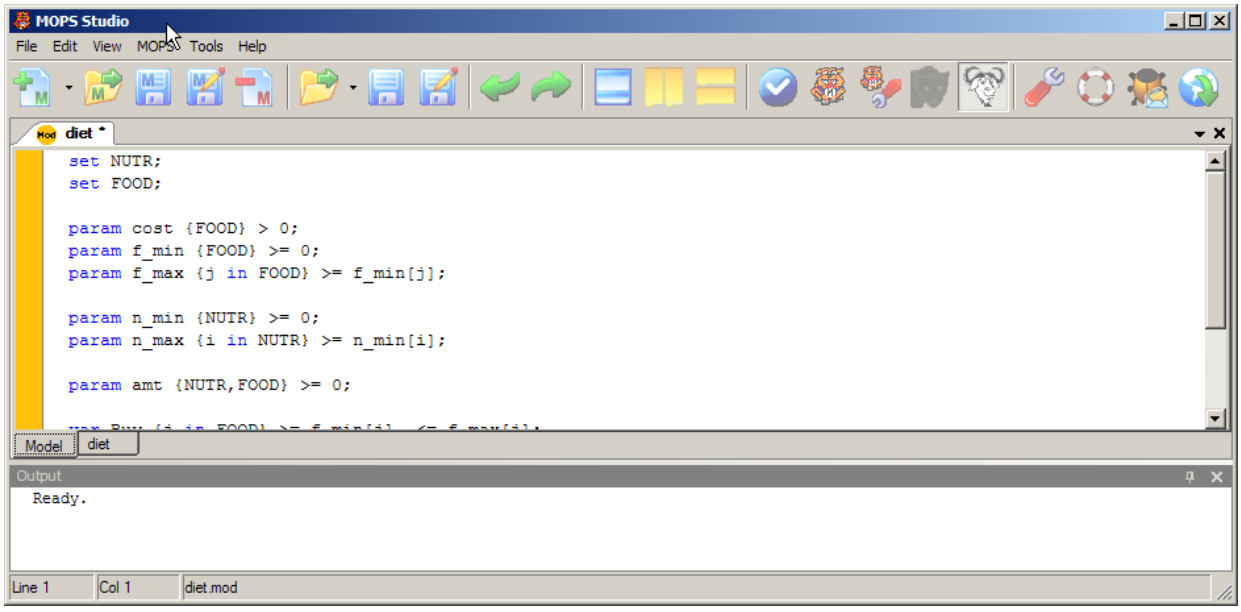


Figure 1. Main Window

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

In Figure 1 you can see the Main Window of MOPS Studio. Here you can find all useful control elements.

Menus

Menus are located across the top of the screen, just below the Title bar. The main menu selections are **File, Edit, View, MOPS, Tools, Window**, and **Help**.
When you choose one of the menus, a submenu drops down to show other options.

Figure 2. Menus bar

- **File** contains commands that apply to the entire model or file such as **New Model**, **Open**, **Save**, and **Close Model.**
- **Edit** contains commands for editing the model or file such as **Undo**, **Redo**, **Cut**, **Copy and Paste.**
- **View** contains commands for controlling the display of the model or file such as **Show and Clear Output Window** and **Show Line Numbers**.
- **MOPS** contains commands as **Check Model Syntax**, **Optimize Model**, **MOPS Parameters and Execute AMPL**
- **Tools** contains functions such as **Convert Model to**, **Rename Variables**, **Write Debug Model File**, **Report Bug** and **Options**.
- **Help** contains links to the Help file and information about the version of MOPS Studio you have installed and automatic online check for updates.

Toolbar

Figure 3. Toolbar

In Figure 3 displays the toolbar of MOPS Studio. It is responsible for the fundamental functions.

- for **creating** a new model. Freely selectable possibilities are **MPS**, **AMPL**, **Triplet** or **Lindo**.
- to **open** a model.
- to **save** the **model** or to **save** the model **as**.
- to **close** the model.
- to open a **MPS**, **AMPL.Mod**, **AMPL.Dat**, **MPL.Mod**, **MPL.Dat**, **Triplet** or **Lindo** file.

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

- to **save** the **file** or to **save** the file **as**.
- to **undo** or **redo**.
- to **semi-maximize** the model window.
- to **split** the model tab **vertically or horizontally**
- to **check** the **model syntax**.
- to **optimize** the **model**.
- to toggle between AMPL and **GNU MathProg** for AMPL model processing
- to **set** the **MOPS parameters**.
- to **execute** an **AMPL** program**.**
- for the **MOPS Studio settings**.
- to get **help**.
- to **report** a **bug**.
- to check online for **updates**.

Input Window

An AMPL-, MPL-, MPS-, Triplet- or LINDO-Modelfile can be written or updated in the Input Window.

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

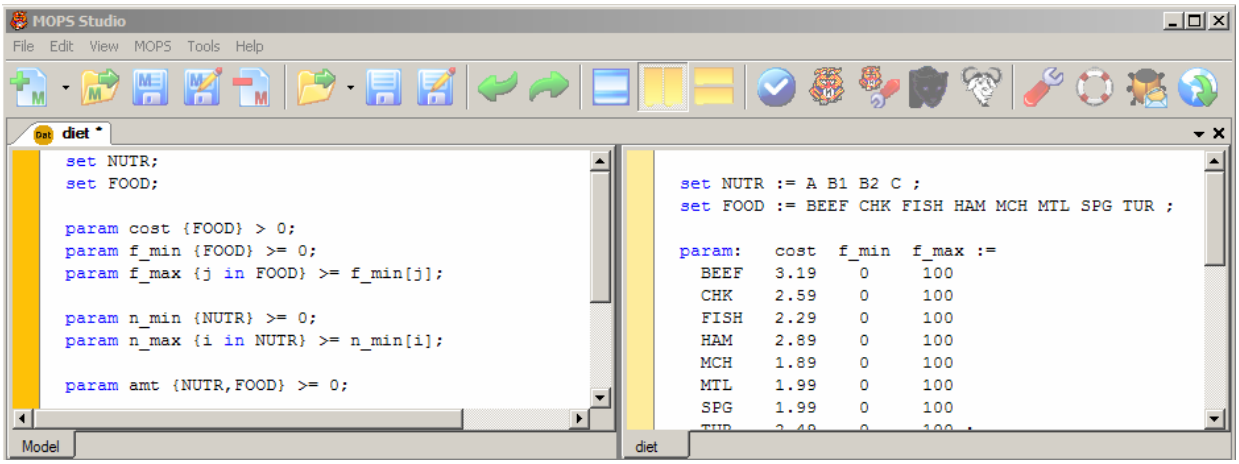

MOPS Studio
File Edit View MOPS Tools Help
diet *
set NUTR;
set FOOD;
param cost {FOOD} > 0;
param f_min {FOOD} >= 0;
param f_max {j in FOOD} >= f_min[j];
param n_min {NUTR} >= 0;
param n_max {i in NUTR} >= n_min[i];
param amt {NUTR,FOOD} >= 0;
Model
set NUTR := A B1 B2 C ;
set FOOD := BEEF CHK FISH HAM MCH MTL SPG TUR ;
param: cost f_min f_max :=
BEEF 3.19 0 100
CHK 2.59 0 100
FISH 2.29 0 100
HAM 2.89 0 100
MCH 1.89 0 100
MTL 1.99 0 100
SPG 1.99 0 100
diet

Figure 4. Input Window split

The Model and Data files of an AMPL-model are stored separately. The entire window can be spilt horizontal or vertical and you can drag and drop files into the windows. Also the tabs can be dragged and dropped.

Output Window

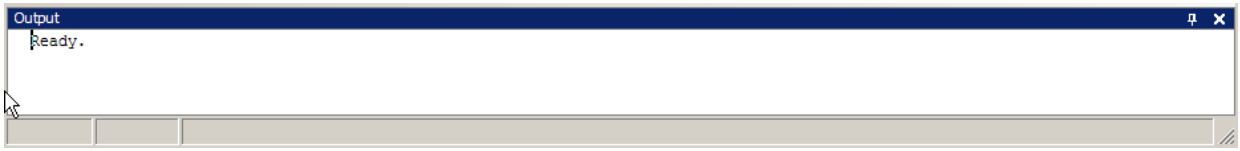


Figure 4. Output Window

The Output Window can be activated with View -> Show Output Window. The Output Window displays all information about the optimization and the syntax errors. The position of a syntax error is marked with the line number and with '>>>' and '<<<' around the error. The relevant line can be found easier if the line numbers are shown (View -> Show Line Numbers).

Like all other windows in the new MOPS Studio versions since v1.7 the output window can be docked and hidden.

Options

In order to change some settings, access **Menu - Tools > Options** or simply click on this button.

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

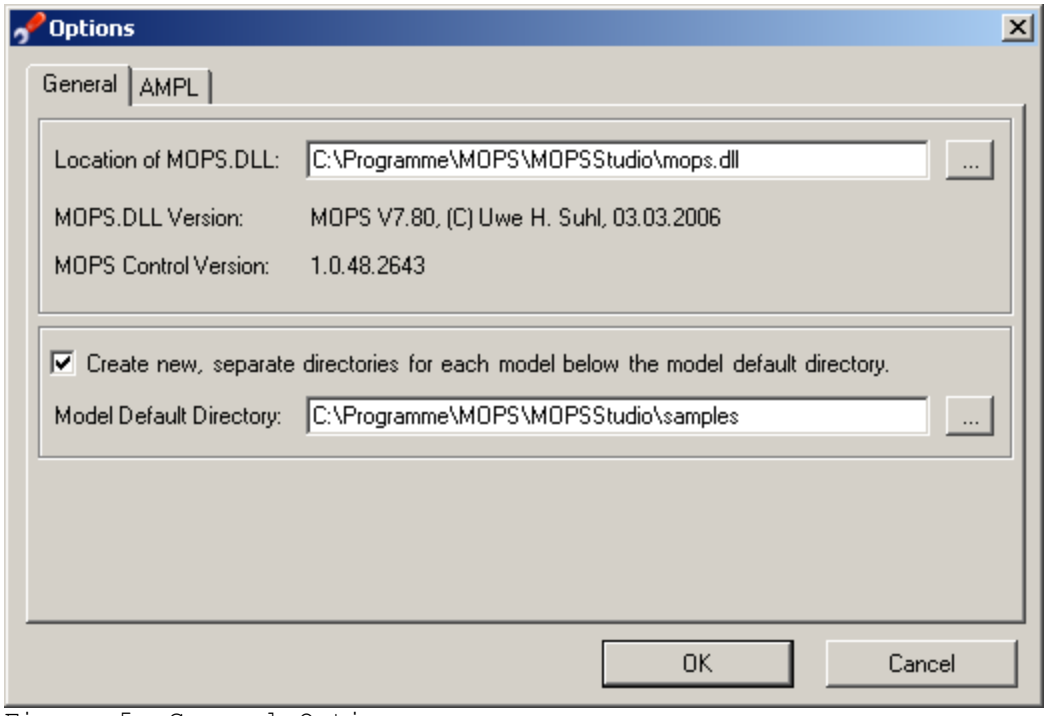


Figure 5. General Options

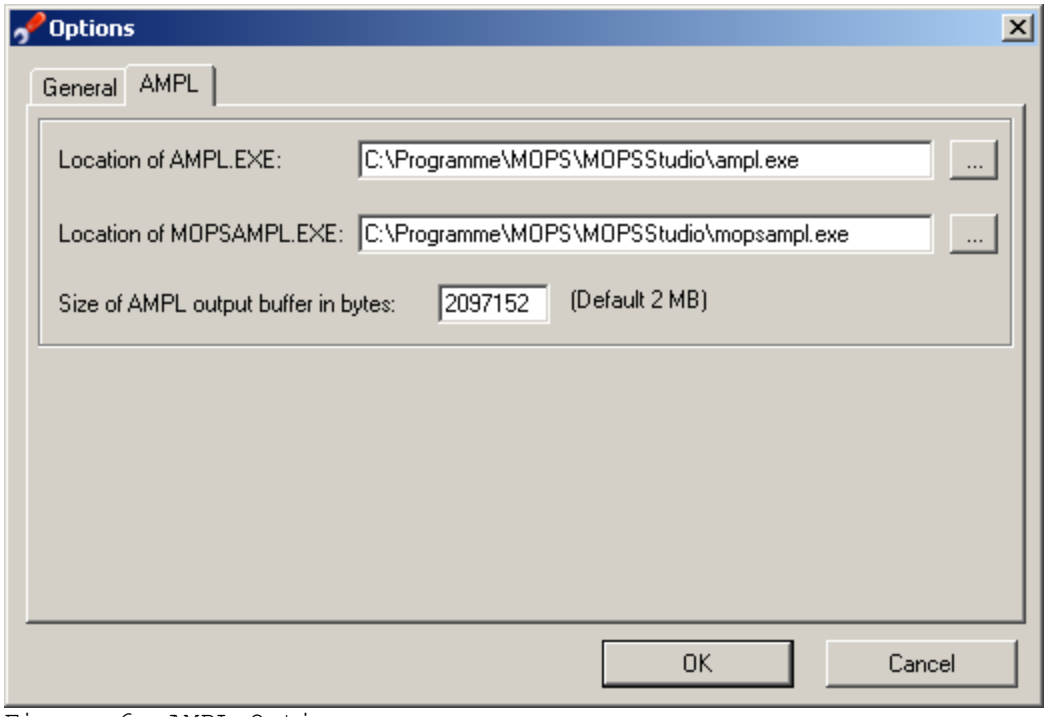


Figure 6. AMPL Options

Parameters

In order to change some parameters, access **Menu - MOPS > MOPS Parameters** or simply click on this button.

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

We recommend you not to change any of this options.

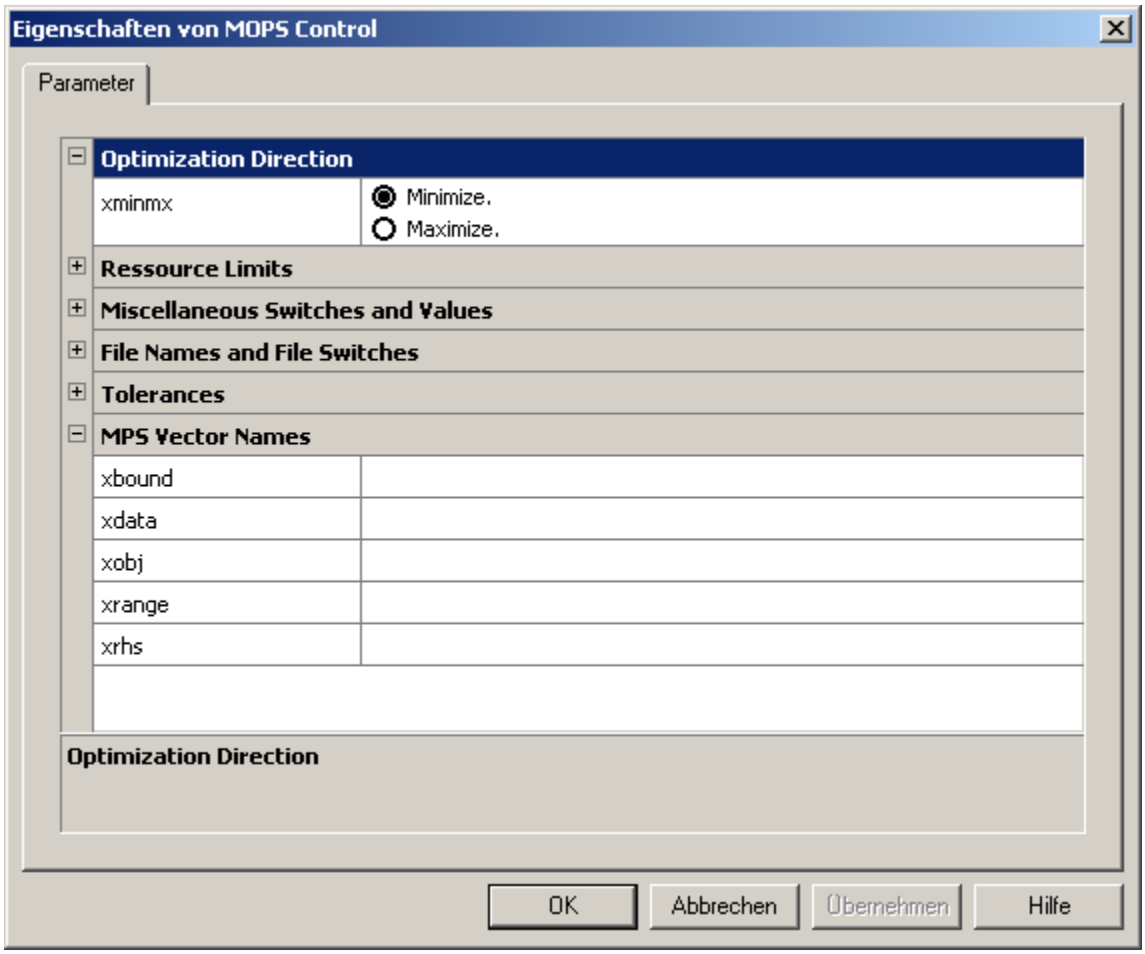


Figure 7. MOPS Parameters

Execute AMPL

The feature Execute AMPL is available in Mops Studio 1.1.5 and above. This feature is necessary to execute AMPL-models with procedural AMPL-commands, i.e. to optimize a slightly updated AMPL-model multiple times in one run. The optimization is started, as soon as the command *solve* is called. Results or additional information are displayed in the Output Window with the command *display*.

The following points are important for using Execute AMPL:

- The whole AMPL output is shown in the Solution Summary window
- All models executed by Execute AMPL cannot be converted to other formats like LINDO, MPS or triplet
- The DAT.file are not included automatically in the AMPL-model. It has to be included with the command *include*.
- If the command *solve* is multiple called, the LP/IP-solution and the static are only refer to the last optimization pass.
- If the execution of an AMPL-model produces a great amount of output, the output buffer has to be increased (see Options). For the most models the default size of 2 MB has is adequate.

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

- If the optimization has to be terminated before the end is reached, the two processes 'ampl.exe' and 'mopsampl.exe' have to be stopped with the task-manager.

Solution

After the optimization of a model, MOPS Studio shows following output files:

- **Statistic**, which includes all information about the model and the optimization.
- **LP-Solution** and -if existing- an **IP-Solution**
- **Solution Summary**, which includes solution values of each variable in the model. It is only shown, when AMPL-models are executed.

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

Chapter VI

MOPS Parameter

6 MOPS Parameter

6.1 MOPS Parameter

MOPS uses a number of parameters. The most important parameters are described in this manual.

- Input Parameter
- Output Parameter

6.2 Input Parameter

6.2.1 Input Parameter

MOPS Input Parameters can be set directly when using the static library, via *SetParameter* when using the DLL or by setting them in a MOPS profile.

Most models can be solved with the default values of MOPS parameters, but very often appropriate parameter tuning reduces the runtime of a model dramatically. For further information refer to the MOPS White Paper.

The input parameter are divided into

- File names
- IP Parameters
- LP Parameters
- MPS vector names
- Ressource limits
- Tolerances
- Other Parameters

6.2.2 File names

Index

xfnbai	xfnbao	xfnips	xfnlog
xfnlps	xfnmps	xfnmsg	xfnpro
xfnsav	xfnsta		

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

xfnbai

Specifies a filename of a basis saved in punch-format from a previous run to restart the LP-optimization with a simplex engine (primal or dual but not IPM).

Type
String *255

See also
xstart

Back to index

xfnbao

Specifies a filename of a basis to be saved saved in punch-format at the end or after xfrbas iterations when using one of the simplex engines (primal or dual).

Type
String *255

See also
xfrbas, xbatyp

Back to index

xfnips

File name (optionally with path) for the integer solution. An IP solution file will be written if *xoutsl > 0.*

Type
String *255

See also
xoutsl

Back to index

xfnlog

File name (optionally with path) for the log file. Parameter *xoutlv* determines if a log file will be written and how detailed it is.

Type
String *255

Back to index

xfnlps

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

File name (optionally with path) for the LP solution. Parameter *xoutsl* determines whether a solution file will be written or not.

Type
String *255

Back to index

xfnmps

File name (optionally with path) of the MPS file.

Type
String *255

See also
xdata, xobj, xrhs, xrange, xbound

Back to index

xfnmsg

File name (optionally with path) of the error message file.

Type
String *255

Back to index

xfnpro

File name (optionally with path) of a MOPS profile. A profile can be used to set MOPS parameters.

Default
xmops.pro

Type
String *255

Back to index

xfnsav

Specifies a filename of a basis to be saved saved in internal format at the end or after xfrbas iterations when using one of the simplex engines (primal or dual).

Type
String *255

See also
xfrbas, xbatyp

Back to index

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

xfnsta

File name (optionally with path) of the statistic file. Parameter *xoutlv* determines if a statistics will be written during optimization.

Type
Long

Back to index

6.2.3 LP Parameters

Index

xbatyp	xbndha	xdjscl	xdropf
xdropm	xfrbas	xfrinv	xipxov
xlptyp	xluctr	xoripm	xpreme
xrduce	xstart	xnproc	xipmem
xnwluf	xnwluu	xfiltl	xipdua

xbatyp

If *xfrbas* >= *0*, then *xbatyp* determines in which format the internal basis will be saved.

Values

1	Internal basis format. Basis will be saved to file *xfnsav*.
2	External basis format (MPS format). Basis will be saved to file *xfnbao*.
3	Both, internal and external basis formats will be saved.

Default
1

Type
Long

See also
xfrbas, xfnbao, xfnsav, xstart

Back to index

xbndha

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

Determines how bounds are treated when solvong initial LPs. IPM and primal simplex (psx) needs less restricted bounds. Dual (dsx) needs restricted bounds for best performance. The default is set to 0 if IPM or psx is used and to 1 if dsx is used. Furthermore in case of a pure LP and use of dsx xbndha is set to 2 and an extended bound reduction is executed. This is not the case for IP-models. To execute the extended bound reduction xbndha has to be set to 2.

Values

0	reduced bounds during LP-preprocessing are relaxed at the end of LPP (default when IPM is used)
1	reduced bounds during LP-preprocessing are kept (default id DSX is used)
2:	extended bound reduction is used

Default

0

Type
Long

Back to index

---

xdjscl

Sets scaling of Reduced Costs.

Values

0	No scaling of Reduced Costs in Pricing.
1	Reduced Costs of non-basis variables are scaled (modified Devex).
2	Full Devex.
3	steepest edge pricing

Default

1

Type
Long

Back to index

---

xdropf

An element in a row or column will be set to 0 during the LU Factorization, if it is less or equal *xdropf* multiplied by a value of an element with the largest magnitude of a row or column.

Range

[0 - 1.0E-9]

Default

1.0E-11

Type
Double

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

Back to index

xdropm

A matrix coefficient with a smaller magnitude than *xdropm* will be set to 0.0.

Range
[0.0 - 1.0E-4]

Default
1.0E-7

Type
Double

Back to index

xfiltl

Determines how much fill is tolerated when using the aggregator during LP-preprocessing. If xfiltl is set to a higher value more fill is allowed. As a consequence the preprocessed matrix and its factorizations may have more nonzeros but fewer constraints.

Range
[0.0 - inf[

Default
4.0

Type
Double

See also
xrduce

Back to index

xfrbas

Iteration frequency for saving the current basis in punch format or internal format if one of the simplex engines is used. If the IPM engine is used on optimal basis can be saved after the cross-over if xfrbas ≥ 0. This allows a warm start with one of the simplex engines.

Values

-1	Basis is never saved.
0	Basis is saved after termination.
p	Basis is saved after p iterations (*p >= 0*).

Default
-1

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

Type
Long

See also
xbatyp, xfnbao, xfnsav

Back to index

xfrinv

A new LU-factorization in one of the simplex engines is computed after the value of xfrinv unless there are numerical problems or a significant amount of fill which makes it more efficient to recompute the LU factors.

Range
[0 - maxint]

Default
determined automatically by the size of the model (dual)

Type
Long

See also
xluctr

Back to index

xipmem

Specifies how the memory for the Cholesky factorization of the IPM-engine is allocated. 1 (default) means the memory is allocated dynamically outside of the MOPS memory block. 0 means it is allocated form the MOPS memory block. Note, that the value of xipmem is only relevant if the IPM engine is used to solve the initial LP (xlptyp = 4). Furthermore if xipmen is set to 0 memory has to be allocated explicitly, i.e. xmreal has to specify the amount of megabytes for the total amount of memory allocated including the Cholesky factorization.

Default
1

Type
Long

See also
xmreal

Back to index

xipxov

Values

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

0	No x-over after solving the initial LP with IPM
1	Only primal part of x-over
2	Primal and dual part of x-over and primal simplex to finish
3	Primal and dual part of x-over and dual simplex to finish

Default
2

Type
Long

See also
xlptyp

Back to index

xipdua

specifies a percent value. If the ratio of the number of active rows and columns after LP-preprocessing is larger than xipdua percent then the IPM-engine solves the dual LP

Values

< 0	IPM engine always solves the primal LP
≥ 0	if m / n > xipdua / 100 the dual LP will be solved by the IPM engine

Default
150

Type
Long

See also
xlptyp
[****]
Back to index

xlptyp

xlptyp sets the algorithm used to solve the initial LP.

Values

0	Primal Simplex
2	Dual Simplex
4	Interior point method.

Default
2

Type

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

Long

Back to index

xluctr

A new LU Factorization will be calculated if the number of elements of the current LU Factorization divided by the number of elements of the last Factorization is greater than *xluctr.*

Range
[1.1 - 5.0]

Default
2.0

Type
Double

See also
xfrinv

Back to index

xnproc

Specifies how many processors should be used for the execution of the IPM-engine on a multiprocessor system. For example on an Intel Core Duo architecture xnproc should be set to 2, resulting in a 40% reduction of the CPU-time. Note, that the value of xnproc is only relevant if the IPM engine is used to solve the initial LP (xlptyp = 4).

Note, CP-time will increase if you specify xnproc > 1 and the machine has less than xnproc processors.

Default
1

Type
Long

See also
xlptyp

Back to index

xnwluf

A new LU-factorization uses memory more efficiently. The old LU-factorization of MOPS can be used by specifying xnwluf = 0. The old LU-factorization offers a small advantage on very sparse models.

Default
1

Type
Long

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

Back to index

xnwluu

A new LU-fupdate can be used in conjunction with the new LU-factorization and has the same advantage. The old LU-update of MOPS can be used by specifying xnwluu = 0. The old LU-update offers a small advantage on very sparse models. Note that the new LU-update requires the new LU-factorization. However, the new LU-factorization can be used in conjunction with the old LU-update.

Default

1

Type
Long

Back to index

xoripm

Ordering strategy for the Cholesky factorization using the interior point method (xlptyp = 4)

Values

0	Automatic ordering
1	Minimum degree
2	Minimum local fill in
3	Nested dissection
4	Multi section ordering

Default

0

Type
Long

Back to index

xpreme

Presolve action control for LP-preprocessing. The bits of this value represent turning on or off individual presolve techniques. To get a particular form of the presolve, set xpreme to the sum of the values corresponding to the following presolve methods:

1 : ***Singleton row check***. Singleton rows are replaced by bounds on variables.

2 : ***Singleton column check***. Free (or implied free) singleton columns are eliminated from the problem.

4 : ***Primal feasibility check***. This process evaluates the possible minimum and maximum row

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

values. Based on the results, it may detect redundant rows and fixes variables on their bound.

8 : ***Cheap dual test***. This procedure performs optimality tests based on the signs of the nonzero values of the columns.

16 : ***Dual feasibility check***. The same as the primal but on the dual problem. Bounds on the dual variables are tightened if possible.

32 : ***Primal bound check and relaxation***. This procedure detects hidden free variables in the problem.

64 : ***Searching identical variables***. This procedure detects splitted variables in the problem and replaces them with an appropriate one. The process also performs this check on the dual problem and removes duplicated rows.

128 : ***Doubleton row check***. This procedure generates free variables in doubleton rows. This procedure modifies the bounds of the variables. Therefore final dual slack values may not correspond to the original problem.

256 : ***Aggregator***. This procedure eliminates free variables.

512 : ***Linear dependency check***. This procedure detects linearly dependent rows in the constraint matrix.

1024 : ***Sparser***. This procedure makes the constraint matrix sparser. This procedure performs row transformations therefore may changes the dual values.

2048 : ***Bound restoring***. MOPS restores the original bounds on the variables after the aggregator if this bit is turned on. In such a case MOPS takes slightly more iterations in primal simplex or IPM in general, but in special cases improves numerical stability.

4096 : ***Extended dual test***. This procedure performs additional dual tests which may change the optimal solution. The optimal objective value is not changed. This procedure can therefore be turned on if only the optimal objective function value is important.

Note: If you need an optimal dual solution corresponding to the original optimization problem, you have to turn off methods 128 and 1024.

Range

[0 - 8191]

Default

2047	for pure LP-models
3903	for pure IP-models
3583	for mixed integer models

For IP models with a small amount of 0-1-variables the strategy 2047 may be more efficient.

Type

Long

Back to index

xrduce

Determines if LP-preprocessing is executed.

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

Values

0	No LP preprocessing.
2	LP Preprocessing.

Default

2

Type
Long

See also
xpreme

Back to index

---

xstart

Specifies how the initial basis is created when using one of the simplex engines.

Values

0	The basis will be defined by all logical variables.
1	A crash basis is produced.
2	Re-start from an external basis *xfnbai*
3	Re-start from save basis *xfnsav*

Default

1

Type
Long

See also
xfnbai, xfnsav

Back to index

6.2.4 MPS vector names

Index

xbound	xdata	xobj	xrange	xrhs

---

xbound

Name of vector to be used as Bound Vector in MPS files. If *xbound* is blank, the first vector in the BOUNDS section of an MPS file will be taken.

Type

String *64

See also
xfnmps

Back to index

---

xdata

Sets the name of the data deck to be processed in an MPS file. If *xdata* is empty, the first data deck encountered in the MPS file will be taken, else the data deck specified by *xdata* will be searched for.

Type
String *64

See also
xfnmps

Back to index

---

xobj

Vector name of objective function in MPS file. This line of the file must be of type "N" (unrestricted). If no name is given MOPS takes the first "N"-row in the data deck as objective function.

Type
String *64

See also
xfnmps

Back to index

---

xrange

Name of the Range Vector in MPS files. If *xrange* is empty, the first vector in the RANGES section of the data deck will be chosen.

Type
String *64

See also
xfnmps

Back to index

---

xrhs

Name of the right hand side vector in MPS file. If *xrhs* is empty the first vector in the RHS section of the MPS file will be taken.

Type
String *64

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

See also
xfnmps

Back to index

6.2.5 Ressource limits

Index

xabgap	xadcol	xadnon	xadrow
xglgap	xmiter	xmitip	xmmax
xmreal	xmxdsk	xmxint	xmxmin
xmxnod	xmxpsu	xmxtlp	

---

xabgap

Absolute gap. The Branch and Bound search will be stopped if $|xzlbnd - xzubnd| \leq xabgap$

Default
0.0

Type
Double

See also
xzlbnd, xzubnd

Back to index

---

xadcol

Maximum number of additional columns for extending the LP/IP model.

Default
0

Type
Long

Back to index

---

xadnon

Maximum number of additional nonzeros for LP- and IP-preprocessing.

Default

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

400000

Type
Long

Back to index

---

xadrow

Maximum number of additional constraints for which can be added during IP-preprocessing.

Default
2000

Type
Long

Back to index

---

xglgap

Relative global gap. The Branch-and-Cut search will be stopped if the relative difference between the objective function value of the best integer solution xzubnd and the objective function value of the best waiting node *xzlbnd* is less than *xglgap.* Thus the search is stopped if |xzlbnd - xzubnd| ≤ xglgap * (1 + |xzubnd|).

Range
[0 - xinf [

Default
1.d-4

Type
Double

See also
xrimpr, xabgap, xzlbnd, xzubnd

Back to index

---

xmiter

Iteration limit. Maximum number of simplex iterations to solve the initial LP. Optimization will be stopped after *xmiter* iterations.

Default
20 000 000

Type
Long

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

Back to index

xmitip

Maximum number of simplex iterations per node during Branch-and-Bound process.

Default
30 000

Type
Long

Back to index

xmmax

Maximum number of constraints.

Default
30 000

Type
Long

Back to index

xmreal

Size of the memory block in Megabyte [MB], that MOPS tries to allocate for model(s), solutions and temporary data. If a positive value is specified for mreal this value is taken overriding the automatic allocation. However, a too small value for mreal might result in an error message. There are rare cases where the automatic computation can fail because the Cholesky or the LU-factorization has an extraordinary amount of fill.
Note that there is the possibility to allocate the Choleky factorization in a separate block of memory if the barrier method is used to solve the initial LP depending on the setting of the parameter xipmem.

Range

0	memory is allocated automatically
[64 - 2000]	allocated memory on a 32-Bit platform
[64 - 8000]	allocated memory on a 64 Bit platform

Default
0

Type
Long

See also
xipmem

Back to index

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

xmxdsk

Maximum disk space in MB used for the node file during the Branch-and-Bound / Cut process.
Note: Previous versions of MOPS saved the tree if *xmxdsk* has been reached (and if *xfrtree = 1*), so that the optimization could be continued later on starting with that tree (*xipbeg = 1)* . This functionality is not supported anymore in recent versions.

Range
[0 - xinf]

Default
100 (MB)

Type
Double

See also
xmxnod

Back to index

---

xmxint

The branch-and-cut search is terminated when the value of xmxint is reached independent of any other search limitation such as CP-time (xmxmin), node limit (xmxnod), disk space (xmxdsk) or global gap (xglgap).

Range
[0 - maxint]

Default
maxint

Type
Long

Back to index

---

xmxmin

Maximum CPU time in minutes for the initial LP and the Branch-and-Bound process. The IP optimization will be stopped if *xmxmin* has been reached.

Note: Previous versions of MOPS saved the B&B tree if *xfrtre >= 0*, so that a restart of the IP optimization was possible. Current MOPS versions do not support this functionality anymore.

Range
[0 - xinf]

Default

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

120

Type
Double

See also
xmxnod, xfrtre , xipbeg

Back to index

xmxnod

Maximum number of nodes in Branch-and-Bound process. The IP optimization will be stopped after *xmxnod* nodes.

Note: Previous versions of MOPS saved the B&B tree if *xfrtre* >= *0*, so that a restart of the IP optimization was possible. Current MOPS versions do not support this functionality anymore.

Range
[0 - maxint]

Default
9 999 999

Type
Long

See also
xmxmin, xipbeg

Back to index

xmxpsu

Max number of passes in IP-preprocessing for improving the strength of the LP Relaxation. The IP-preprocessing will be terminated after *xmxpsu* passes or if the objective function value does not change anymore.

Range
[0 - maxint]

Default
10

Type
Long

Back to index

xmxtlp

Maximum CPU time in minutes for solving the initial LP. The optimization will be terminated after *xmxtlp*

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

minutes if no solution has been found until then.

Range
[0 - xinf]

Default
xinf

Type
Double

See also
xmxnod, xfrbas, xbatyp

Back to index

6.2.6 Tolerances

Index

xdropm	xrimpr	xtold1	xtold2
xtolin	xtolpv	xtolqi	xtolr1
xtolr2	xtolre	xtolx	xtolx1
xtolx2	xtolzr		

---

xdropm

A matrix coefficient smaller than *xdropm* will be set to 0.0.

Range
[0.0 - 1.0E-4]

Default
1.0E-7

Type
Double

Back to index

---

xrimpr

After each IP solution found, *xzubnd* will be set to *xzbest - xrimpr * |xzbest| * xsscal,* so the next integer solution must be *xrimpr * 100%* better than the previously found solution with objective function value *xzbest.* After the Branch-and-Bound process it will be proved, that the best solution found is maximal *xrimpr * 100%* away from the global optimum. *xsscal = 1* when minimizing and *xsscal = -1* when maximizing.

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

Range
[0.0 - 1.0]

Default
1.0E-4

Type
Double
Back to index

xtold1

Dual feasibility tolerance in phase 1 of the Simplex. Reduced costs must be greater than *xtold1* to be taken into the basis.

Range
[1.0E-10 - 1.0E-1]

Default
1.0E-7

Type
Double

Back to index

xtold2

Dual feasibility tolerance in phase 2 of the Simplex. Reduced costs must be greater than *xtold2* to be taken into the basis.

Range
[1.0E-10 - 1.0E-1]

Default
1.0E-7

Type
Double

Back to index

xtolin

A solution value will be regarded as integer if it is not more than *xtolin* away from the next integer value.

Range
[1.0E-3 - 1.0E-6]

Default

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

1.0E-5

Type
Double

Back to index

xtolpv

Minimum pivot value in Simplex. Tolerance for the determination of the pivot element.

Range
[1.0E-8 - 1.0E-3]

Default
1.0E-7

Type
Double

Back to index

xtolqi

A solution value will be regarded as quasi integer (to the controlling of the Branch&Bound-Heuristic) if it is not more than *xtolqi* away from the next integer value.

Range
[1.0E-3 - 1.0E-6]

Default
0.05

Type
Double

See also
xheutp

Back to index

xtolr1

Primal relative feasibility tolerance in phase 1 of the Simplex. A variable *x* with lower bound *lb* and upper bound *ub* will only be regarded as feasible if

*lb - xtolx1 - xtolr1 * |lb| <= x <= ub + xtolx1 + xtolr1 * |ub|*

This rule is also applied to logical variables.

Range
[0.0 - 1.0E-7]

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

Default
1.0E-12

Type
Double

See also
xtolx1

Back to index

xtolr2

Primal relative feasibility tolerance in phase 1 of the Simplex. A variable *x* with lower bound *lb* and upper bound *ub* will only be regarded as feasible if

*lb - xtolx2 - xtolr2 * |lb| <= x <= ub + xtolx2 + xtolr2 * |ub|*

This rule is also applied to logical variables.

Range
[0.0 - 1.0E-7]

Default
1.0E-14

Type
Double

See also
xtolx2

Back to index

xtolre

Primal relative feasibility tolerance after termination of the LP run. A variable *x* with lower bound *lb* and upper bound *ub* will only be regarded as feasible if

*lb - xtolx - xtolre * |lb| <= x <= ub + xtolx + xtolre * |ub|*

This rule is also applied to logical variables.

Range
[0.0 - 1.0E-7]

Default
1.0E-10

Type
Double

See also

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

xtolx

Back to index

xtolx

Primal absolute feasibility tolerance after termination of the LP run. A variable *x* with lower bound *lb* and upper bound *ub* will only be regarded as feasible if

*lb - xtolx - xtolre * |lb| <= x <= ub + xtolx + xtolre * |ub|*

This rule is also applied to logical variables.

Range
[1.0E-9 - 1.0E-3]

Default
1.0E-4

Type
Double

See also
xtolre

Back to index

xtolx1

Primal absolute feasibility tolerance in phase 1 of the Simplex. A variable *x* with lower bound *lb* and upper bound *ub* will only be regarded as feasible if

*lb - xtolx1 - xtolr1 * |lb| <= x <= ub + xtolx1 + xtolr1 * |ub|*

This rule is also applied to logical variables.

Range
[1.0E-9 - 1.0E-3]

Default
1.0E-6

Type
Double

See also
xtolr1

Back to index

xtolx2

Primal absolute feasibility tolerance in phase 2 of the Simplex. A variable *x* with lower bound *lb* and

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

upper bound *ub* will only be regarded as feasible if

*lb - xtolx2 - xtolr2 * |lb| <= x <= ub + x tolx2 + xtolr2 * |ub|*

This rule is also applied to logical variables.

Range
[1.0E-9 - 1.0E-3]

Default
1.0E-5

Type
Double

See also
xtolr2

Back to index

---

xtolzr

The result of a floating point calculation will be set to zero if its absolute value is smaller than *xtolzr.*

Range
[1.0E-13 - 1.0E-9]

Default
1.0E-12

Type
Double

Back to index

6.2.7 Other Parameters

Index

xchksw	xdimcn	xfrlog	ximrrw
xinf	xinfor	xminmx	xoutlv
xoutsl	xmaxel	xrcmax	xscale
xstorn	xwrmps	xnwcon	

---

xchksw

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

Determines if a model will be checked syntactically before optimization.

Values

0	No syntax check.
1	Checking if the model is syntactically correct.

Default

0

Type
Long

Back to index

xdimcn

Sets the maximum model dimensions used when reading MPS files.

Values

0	Set the maximal model dimensions specified by *xnmax, xmmax, xnzmax*.
1	MPS Convert sets the max dimensions automatically.

Default

1

Type
Long

Back to index

xfrlog

Sets LP logging frequency. After *xfrlog* LP Simplex iterations a line will be written to the log file, if *xoutlv = 3*.
Logging is very time-consuming and should only be activated during debugging.

Default

0

Type
Long

See also
xoutlv

Back to index

ximrrw

If *ximrrw* is 1, the model in main memory will be written to the hard disk and retrieved after the optimization run. If you would like to make changes to the original model or read model information after

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

the optimization, for example to modify the model for a new optimization run, you have to activate this parameter.

Values

0	No model will be saved before optimization
1	Model will be saved before optimization for later changes

Default

0

Type
Long

Back to index

xinf

Represents an infinite value. This value is used for specifying infinite bounds on variables and constraints. xinf is an internal parameter and cannot be changed in profiles or via *SetParameter.*

Range

(1.0E10 - 1.0E30)

Default

1.0E20

Type
Double

Back to index

xinfor

Sets input format of model data.

Values

0	Internal format
1	MPS format
2	Triplet format

Default

1

Type
Long

See also
xfnmps

Back to index

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

xminmx

Sets optimization direction.

Values

min	Model is minimized.
max	Model is maximized.

Default

min

Type
String *8

Back to index

xnwcon

If the mps-input model is in free format and has long names for the identifiers then xnwcon must be set to 1. Note that the convert time is significantly longer if the new convert is used for traditional mps-files in fixed format. In such a case xstorn is set automatically to 1 and the names are stored in additional arrays.

Values

0	the standard convert (according to the IBM specification) is used, i.e. max 8 characters, fixed format, embedded, leading and trailing blanks are significant
1	free format, max. 255 characters for identifiers, no imbedded blanks, leading and trailing blanks are not significant.

Default

0

Type
Integer*4

Back to index

xoutlv

Output level for logging. Controls creation of .log and .sta files.

Values

0	No logging, no statistics.
1	Write statistic file after optimization, no log file.
2	Write statistic file and log file.
3	Write statistic file and log file with extended information. After each *xfrlog* iterations a information will be written to the log file.

Default

2

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

Type
Long

See also
xfnsta, xfrlog

Back to index

xoutsl

Determines if IP / LP solution will be written to solution file(s).

Values

0	No output to solution file.
1	only the column information of an LP / IP solution will be written to a file..
2	row and column information of an LP / IP solution will be written to a file

Default
1

Type
Long

See also
xfnlps, xfnips

Back to index

xmaxel

Maximum value of matrix coefficient. If it is greater than *xmaxel*, an error will occur. *xmaxel* cannot be changed in profiles or via *SetParameter.*

Default
1.0E-5 * xinf

Type
Double

See also
xinf

Back to index

xrcmax

Determines how the size of the character space for row and column names is computed. Note if standard mps-files are used there is no need to store names. In case of the convert based on the free format xrcmax is computed automatically based on the problem size.

Values

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

0	no storage of names
1	(xn + xm) * xdfnal
> 1	this value in bytes

Default
0

Type
Long

See also
xstorn

Back to index

xscale

Determines if the model is scaled before optimization.

Values

0	No scaling.
1	Row-wise scaling.
2	Row-wise and column-wise scaling.

Default
2

Type
Long

Back to index

xstorn

Determines if row and column names are to be stored.

Values

0	No storage of names.
1	Storage of names from mps-data and/or in the context of DLL-functions.

Default
1

Type
Long

See also
xrcmax

Back to index

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

xwrmps

Determines whether an MPS-file is written from the internal model representation (before LP-preprocessing)

Values

0	mps-file is not written.
1	mps-file is written before LP-preprocessing.

Default

0

Type
Long

Back to index

6.2.8 IP Parameters

Index

xadrow	xbckpa	xbndha	xbndrd	xbrheu
xclict	xcored	xcovct	xeucrd	xflwct
xflwpa	xfrtre	xfrnod	xgomct	xgommc
xheutp	xhfinb	xhgap	xhrdlb	xhrdub
xhtlim	ximbnd	ximpli	xipbnd	xiplpt
xlifo	xlocs	xlotst	xlpgap	xlpmip
xmnheu	xmirct	xnewbb	xnlbab	xnlstr
xnodbf	xnodse	xnoitb	xnopre	xnogap
xparnd	xpcini	xprlev	xranha	xrimpr
xusepl				

xadrow

Maximum number of additional constraints which can be added during IP-preprocessing.

Default

2000

Type
Long

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

Back to index

xbckpa

Backtracking parameter for mixed lifo node selection strategy (xnodse<0 and xlifo=2)

Range
[0.0,1.0]

Default
0.999

Type
Double

See also
xnodse, xlifo

Back to index

xbndha

The dual simplex engine (xlptyp = 2) benefits from tight bounds derived during the LP-preprocessing (LPP). xbndha determines the level of bound reduction and bound retaining after LP-preprocessing.

Values

0	Level 0: tight bounds from LPP are retained if the initial LP is solved by the dual simplex and the model is a pure LP-model
1	Level 1: as 0 but in addition an extended bound reduction is used to further tighten the bounds
2	Level 2: as 1 but the extended bound reduction is also executed when the model is an IP-model

Default
0

Type
Long

Back to index

xbndrd

Level of bound reduction during IP-processing and branch-and-bound / cut

Values

0	Level 0: no bound reduction
1	Level 1: bound reduction of all variables during IP-processing and branch-and-bound / cut
2	Level 2: extended bound reduction during IP-processing and branch-and-bound

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

	/ cut

Default

1

Type
Long

Back to index

---

xbrheu

Branching heuristic. Determines how a branching variable will be chosen.

Values

0	When minimizing (maximizing) a fractional IP variable with maximal (minimal) objective function coefficient will be set to the down rounded value.
1	When minimizing (maximizing) a fractional IP variable with minimal (maximal) objective function coefficient will be set to the up rounded value.
2	Branch on most fractional IP-variable. Tie breaking rule is the best cost.
3	Branch on IP-variable with largest/smallest costs if minimization/maximization. The variable will be set to the down rounded value. Tie breaking rule is most fractional part.
5	Branch on fractional integer variable with largest weighted pseudo costs. Tie breaking rule is most fractional part. If global gap is less than 0.02 change to xbrheu=3.
6	Branch on fractional integer variable with the largest smallest pseudo costs. Tie breaking rule is most fractional part

Default

5

Type
Long

Back to index

---

xclict

Determines if Cliques Cuts will be produced during IP-preprocessing.

Values

0	No Cliques will be derived.
1	All Cliques will be saved and used during logical tests.
2	Additionally to option 1, violated Clique Cuts are added during IP-preprocessing.

Default

2

Type
Long

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

Back to index

xcored

Sets coefficient reduction level during IP-preprocessing

Values

0	No coefficient reduction.
1	Simple coefficient reduction.
2	Extended coefficient reduction, based on Probing (can be time-consuming).

Default

1

Type
Long

Back to index

xcovct

Determines if and how Cover Cuts are derived during IP-preprocessing.

Values

0	No Cover Cuts are derived.
1	Violated Cover Cuts are derived only during the initial IP-preprocessing.
2	Cover Cuts are only derived when the objective function value improves.
3	Cover Cuts are derived during each IP-preprocessing.

Default

2

Type
Long

Back to index

xeucrd

Euclidean reduction of constraints during IP-processing.

Values

0	no euclidean reduction
1	use euclidean reduction

Default

1

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

Type
Long

Back to index

xflwct

This parameter determines if flow cuts should be derived

Values

0	flow cover cuts are not used
1	Simple generalized flow cover inequalities (SGFCIs)
2	Extended generalized flow cover inequalities (EGFCIs)
3	more violated of SGFCI and EGFCI
4	Lifted simple generalized flow cover inequalities (LSGFCIs)
5	Lifted flow cover inequalities (LFCIs)
6	more violated of LSGFCI and LFCI

Default
4

Type
Long

Back to index

xflwpa

This parameter determines if flow path cuts should be derived

Values

0	flow path cuts are not used .
1	simple flow path cuts .
2	extended flow path cuts .

Default
0

Default
0

Type
Long

Back to index

xfrnod

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

Determines if the branch-and-bound / cut process of xoptim / Optimize will be interrupted if xfrnod ≥ 0.
Usage example in pseudo code:

```
do
      call xoptim (dir, status, phase, funct) or optimize (dir, status, phase, funct)
      if  xrtcod .=  5 then          ! interrupted after xfrnod nodes or integer solution
         write '  IP-value ', funct
         cycle
      else if  xrtcod .>  0  then  ! fatal error - exit
         exit
      else                              ! normal exit
         exit
      end if
enddo
```

Values

-1	no interrupt of the branch & bound / cut process
0	an interrupt occurs after an automatically computed number of nodes or an integer solution was found
> 0	an interrupt occurs after xfrnod nodes or an integer solution was found

Default

-1

Type
Long

Back to index

---

xfrtre

Re-starting an IP optimization from a saved branch-and-bound tree is a discontinued function, not supported by recent MOPS versions!

Determines if the Branch-and-Bound tree will be saved for posterior re-starts when reaching a resource limit.

Values

0	B&B tree will not be saved.
1	B&B-tree will be saved to file.

Default

0

Type
Long

See also
xmxnod, xmxmin, xipbeg

Back to index

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

xgomct

Determines Determines if and how Gomory Mixed-Integer Cuts will be derived during IP-preprocessing.

Values

0	No Cuts will be derived.
1	Cuts with a limited number of nonzero elements are derived.
>1	The higher the value (max 4) the more cuts will be derived. Note, that cuts are relatively dense, which slows down the LP-optimization at nodes.

Default

2

Type
Long

Back to index

---

xgommc

Sets the density level for taking Gomory cuts into the matrix. A violated Gomory cut will only be added to the matrix if the sum of all nonzeros in all columns of the cut is smaller than *xgommc * xn.* A smaller value of *xgommc* results in a smaller number of Gomory cuts and possibly weaker LP-relaxation, but faster LP-reoptimization..

Default

6

Type
Long

Back to index

---

xheutp

Determines the IP heuristic to be used to obtain initial IP-solutions.

Values

0	No heuristic is used.
1	Total rounding: integer variables will be rounded to the nearest integer value in several passes.
2	Local Branching
3	Relaxation Induced Neighborhood Search (RINS) heuristic
4	Relaxation-based Search Space (RSS) heuristic with LIFO. If xhfinb =1 a part of the nonbasic variables will be fixed and all basic integer variables with a fractional part less than xhrdlb will be fixed to the next lower integer value. Basic integer variables with a fractional part larger than xhrdub will be fixed to the next larger integer value.
5	RSS heuristic with xnodse =3
6	RSS heuristic with xnodse =3 and total rounding xlocs=4 every xnlbab=5 nodes.

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

Default
6

Type
Long

See also
xmnheu, xhtlim, xhgap

Back to index

xhfinb

Determine whether a subset of the nonbasic integer variables are fixed before the heuristic will start. This parameter is only valid for Local Branching-, RINS- and RSS-heuristic (xheutp > 1).

Values

0	no none basics are fixed
1	a subset of nonbasic integer variables are fixed before the heuristic will start

Default
1

Type
Long

See also
xheutp

Back to index

xhgap

If the global relative gap in the heuristic is smaller than *xhgap*, the heuristic will be terminated. This parameter is only relevant, if a heuristic is used (xheutp > *0)*.

Default
0.05

Type
Double

See also
xheutp, xmnheu, xhtlim

Back to index

xhrdlb

A fractional integer variable will be rounded down, if its fractional part is smaller than *xhrdlb*. This parameter is only relevant for the RINS- and RSS-heuristic (xheutp ≥ 3).

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

Range

$]0.0-1.0[$

Default

depends on heuristic and size of the problem

Type
Double

See also
xheutp, xhrdub

Back to index

xhrdub

A fractional integer variable will be rounded up, if its fractional part is larger than *xhrdub*. This parameter is only relevant for the RSS-heuristic (xheutp $\geq$ 4).

Range

$]0.0-1.0[$

Default

depends on heuristic and size of the problem

Type
Double

See also
xheutp, xhrdlb

Back to index

xhtlim

Time limit in the heuristic. This parameter is only relevant if a heuristic is used (xheutp > *0)*.

Range

[0 - xmxmin]

Default

10.d0

Type
Double

See also
xhgap, xheutp, xmnheu

Back to index

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

ximbnd

Determines the use of conditional bound implications (only mixed integer models) during IP-preprocessing and branch and bound / cut.

Values

0	Bound Implications are not derived and not used
1	Bound Implications are only used during IP-preprocessing.
2	Bound implications are also used during the branch and bound / cut.

Default

2

Type
Long

Back to index

ximpli

Determines how implications, that have been derived by probing, will be used during IP-preprocessing.

Values

0	No implications will be derived.
1	All implications are derived and stored, they will be used in all logical tests.
2	Additionally to option 1 violated Implication Cuts are added during IP-preprocessing.

Default

2

Type
Long

Back to index

xipbnd

The user can specify a tree bound to limit the search: a node with a LP-value worse than *xipbnd will be dropped.*

Range

]-xinf - +xinf[

Default

xinf	When minimizing
-xinf	When maximizing

Type
Double

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

See also
xrimpr, xlpgap, xglgap

Back to index

xiplpt

Sets the algorithm to be used to solve the LP at each node during the branch-and-bound process.

Values

0	Primal Simplex
> 0	Dual Simplex.

Default

1

Type
Long

See also
xlptyp

Back to index

xlifo

Set type of mixed LIFO. Mixed LIFO is activated if xnodse<0.

Values

0	end LIFO, if node limit is reached
1	end LIFO, if LP infeasible or integer
2	end LIFO, if functional value is larger than the value: |xzubnd-xzlbnd|*xbckpa

Default

0

Type
Long

See also
xnodse, xnlbab, xnlstr

Back to index

xlocs

Sets thetype of local search to be used: the neighborhood of a node is searched for integer solutions. Note that local search can be costly.

Values

0	no local search

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

1	relaxation induced neighborhood search (RINS)
2	local branching
3	rounding in branch and bound
4	total rounding at every xnlbab nodes
5	various rounding and diving heuristics at every xnlbab nodes

Default

0

Type
Long

See also
xnodse, xnlbab, xnlstr

Back to index

xlotst

Determines the level of bound reduction for continuous variables.

0	No bound reduction of continuous variables
1	all bounds of continuous variables are reduced
2	only the bounds of continuous variables with free integer variables in its rows are reduced
3	only the bounds of continuous variables with a change greater equal 0.05 are reduced

Default

2

Type
Long

Back to index

xlpgap

if *xipbnd* = |*xinf*| *and* *xlpgap* < *xinf*, a tree bound xipbnd is computed after the first IP-preprocessing: *xipbnd = funct* + *xlpgap* * |*funct*| if minimization and *xipbnd = funct - xlpgap* * |*funct*| *if maximization. Note that a too small value* for *xlpgap* might lead to a branch-and-bound / cut search with no integer solution..

Range

[0 - xinf]

Default

xinf

Type
Double

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

See also
xipbnd, xinf

Back to index

xlpmip

This parameter determines if an IP model will be solved as IP or if only its LP-relaxation is solved. Note that the LP-preprocessing might be different in both cases.

Values

0	Only the LP-relaxation of an IP model will be solved.
1	IP-model will be solved.

Default
1

Type
Long

Back to index

xmirct

This parameter determines if mixed integer rounding cuts should be derived during IP-preprocessing..

Values

0	MIR-cuts are not used
1	MIR-cuts are derived only at the first IP-preprocessing and only from original constraints.
2	MIR-cuts are derived at every IP-preprocessing but only from original constraints.
3	MIR-cuts are derived at every IP-preprocessing and all active constraints are considered.

Default
3

Type
Long

Back to index

xmnheu

Sets number of nodes to be processed by the heuristics before the branch-and-bound process. This parameter is only relevant if heuristics are used (*xheutp > 0).*

Range
[0 - maxint]

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

Default
50

Type
Long

See also
xheutp

Back to index

---

xnewbb

This parameter allows to activate the old branch-and-bound algorithm..

Values

0	old branch and bound
1	new branch and bound

Default
1

Type
Long

Back to index

---

xnlbab

Node limit in branch and bound before the next local search or mixed LIFO iteration starts. This parameter is relevant if xnodse < 0 or xlocs > 0.

Range
[0 - xmxnod]

Default
1000

Type
Long

See also
xnodse, xlocs, xnlstr, xlifo

Back to index

---

xnlstr

Node limit during local search heuristic or mixed lifo strategy. This parameter is only relevant if xnodse < 0 or xlocs > 0.

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

Range
[0 - xmxnod]

Default
1000

Type
Long

See also
xnodse, xlocs, xnlbab, xlifo

Back to index

xnodbf

Combine a node selection strategy (xnodse>1) with best first selection.

Values

-1	no best first selection
0	the frequency of the best first strategy is computed depending on the gap
x>0	best first strategy is used every x node

Default
0

Type
Long

See also
xnodse

Back to index

xnodse

Sets the node selection strategy in the branch-and-bound process.

<0	mixed LIFO strategy with a node selection |xnodse|
0	LIFO
1	Chose node with best objective function coefficient.
2	Chose node with minimum sum of integer infeasibilities.
3	Best Projection, with xnodse = *2* until the first IP solution.
4	Best Projection (original version).
5	Chose node with best estimation based on pseudo costs
6	Chose node until first IP solution with xnodse=2, than with xnodse=5 until gap is smaller than 5% and finally chose node with percentage error
7	Best estimation based on pseudo costs, with xnodse=2 until first IP solution

Default

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

3

Type
Long

See also
xlifo, xnodbf

Back to index

xnogap

xrimpr and xglgap are set to default values of 1.d-4. If the IP-model has very large objective function values then it is possible that a true optimal IP-solution may nit be found because xrimpr and xglgap are relative values. Therefore we compute rimpr=min(xrimpr,1/(1+|zlb|) and glgap=min(xglgap,1/ (1+ |zlb|), where zlb is the functional value after IP-preprocessing. If one the values rimpr or glgap are smaller then the corresponding value of xrimpr or xglgap the value is replaced.

Values

0	dynamic computation of xrimpr and xglgap after IP-preprocessing and possible reduction of these values if the LP-objective function zlb is sufficiently large
1	no dynamic computation; xrimpr and xglgap are unchanged

Default

0

Type
Long

See also
xrimpr, xglgap

Back to index

xnoitb

Indicate whether integer table is build. The integer table is automatically built if extended integer types are present. In all other cases there is no need to built an integer table where various information about integer variables are stored.

Values

0	integer table can be build
1	no integer table

Default

1

Type
Long

Back to index

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

xnopre

Allows to turn off IP-preprocessing.

Values

0	no IP-preprocessing
> 0	IP-preprocessing with individual settings

Default

1

Type
Long

Back to index

xparnd

Set the partitioning type for the node table. The node table is partitioned in two sets of nodes: those which are immediate candidates and those which are waiting. If the IP-model still has a bad LP-relaxation after IP-preprocessing xparnd should be set to zero, because the search in the first set most likely results in no integer solution. As a consequence the search process for an initial integer solution takes longer than necessary.

Values

0	no partitioning
1	partition node table

Default

1

Type
Long

Back to index

xpcini

Set the initialization type for the computation of pseudo costs..

Values

0	initialize pseudo costs to zero
1	initialize pseudo costs to variable cost
2	solve LP partially to compute pseudo costs (not activated)

Default

1

Type
Long

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

See also
xnodse, xbrheu, xparnd

Back to index

xprlev

Probing level. Probing tries to set binary variables, that are not fixed, to 0 or 1 on a tentative basis and analyzes the resulting implications. In pure 0/1 models this technique can be very effective. For mixed integer models, probing is usually less effective.

Values

0	Probing only on fractional 0-1-variables during IP-preprocessing.
1	Probing on all 0-1-variables, but only at IP-preprocessing
2	Probing on all 0-1-variables at IP-preprocessing and at each node of the tree.

Default
1

Type
Long

Back to index

xranha

Sets conversion of Range constraints that contain integer variables.

Values

0	No conversion.
1	Range constraints with integer variables are converted into two separate constraints to allow a better IP-preprocessing.

Default
0

Type
Long

Back to index

xrimpr

This value determines the relative improvement required for the next integer solution.

Range
[0 - xinf [

Default
1.d-4

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

Type
Double

See also
xglgap, xabgap, xnogap

Back to index

xusepl

This value determines if a cut pool is derived and stored. Note that a cut pool requires a significant amount of additional main memory.

Values

0	do not derive a cut pool, i.e. all violated cuts during IP-preprocessing are included into the model.
1	derive a cut pool: all violated cuts are stored in a cut pool. Finally a subset of the cuts are included into the model according some sophisticated selection algorithms

Default
0

Type
Integer

Back to index

6.3 Output Parameters

6.3.1 Output Parameters

The following MOPS parameters can be retrieved with *GetParameter()* when using MOPS.DLL or directly when using the IMR interface.

Index

xcmgap	xctime	xertyp	xipfun
xipsta	xiptim	xiter	xline1
xline2	xlpfun	xlpsta	xlptim
xm	xmorig	xmxj	xmxj1
xn	xnints	xnodes	xnzero
xrtcod	xversn	xzlbnd	xzubnd

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

xcmgap

Global gap in % if IP optimization has been stopped before an optimal solution was found.

Type
Double

Back to index

xctime

Total optimization time in minutes.

Type
Double

Back to index

xertyp

Internal error number.

Type
Long

See also
Error Message

Back to index

xipfun

Objective function value of the best IP solutions found so far.

Type
Double

Back to index

xipsta

IP optimization status:

0	Branch & Bound search terminated
1	Search not yet terminated

Type
Long

Back to index

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

xiptim

IP optimization time in minutes (without initial LP).

Type
Double

Back to index

xiter

Current number of Simplex iterations.

Type
Long

Back to index

xline1

First line with context sensitive information (max. 72 characters).

Type
C72

Back to index

xline2

Second line with context sensitive information (max. 72 characters).

Type
C72

Back to index

xlpfun

Objective function value if the initial LP.

Type
Double

Back to index

xlpsta

LP solution status:

0	optimal solution found
1	the problem has no feasible solution
2	unbounded solution

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

Type
Long

Back to index

xlptim

Time consumed for solving the initial LP in seconds

Type
Double

Back to index

xm

Current number of constraints of the internal model.

Type
Long

Back to index

xmorig

Number of constraints in the original model, before LP-preprocessing.

Type
Long

Back to index

xmxj

Current number of integer variables including 0-1-variables

Type
Long

Back to index

xmxj1

Current number of binary variables.

Type
Long

Back to index

xn

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

Current number of structural variables.

Type
Long

Back to index

xnints

Number of integer solutions found so far

Type
Long

Back to index

xnodes

Current number of nodes in Branch & Bound process.

Type
Long

Back to index

xnzero

Current number of nonzero elements in the internal model.

Type
Long

Back to index

xrtcod

Return code of the last procedure executed.

If the optimization of a LP-model has an abnormal ending, *xrtcod* can have following values

1	Iteration or resource limit reached
2	input error in IMR or basis
3	output error
4	numerical problem
6	system error

If the Branch-and-Bound ends abnormal, *xrtcod* can have following values

1	Node, Iteration or CPU-time limit reached
2	input error in IMR or Branch-and-Bound
3	output error

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

4	numerical problem
5	not enough storage for LU matrix
6	system error

Type
Long

See also
xertyp

Back to index

xversn

MOPS version info.

Type
C80

Back to index

xzlbnd

Current lower (upper) bound of all waiting nodes when minimizing (maximizing).

Type
Double

Back to index

xzubnd

Current upper (lower) bound of all waiting nodes when minimizing (maximizing).
xzubnd is calculated from xipfun and xrimpr.

Type
Double

Back to index

6.3.2 Error Message

Every error has an explicit error code, which is saved in the output parameter xertyp. The following table includes the error codes with the corresponding error messages.

error code	error message
0001	LP iteration limit xmiter reached; xmiter is:
0002	LP time limit xmxtlp reached; xmxtlp is:

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

0003	IP time limit xmxmin reached; xmxmin is:
0004	node LP iteration limit xmitip reached; xmitip is:
0005	insufficient space for LU factors - increase xmxnen
0006	node limit xmxnod reached; xmxnod is:
0007	nodes buffer size to small - increase xmxnin
0008	disk full - optimization terminated
0009	granted disk space exhausted - increase xmxdsk
0010	xmxnen must be larger than xnzero
0011	pivot element in LU-factorization is too small
0012	disk capacity insufficient to restart IP-optimization!
0013	node memory insufficient to restart IP-optimization!
0014	required no IP-solutions reached
0015	basis incorrect - basis size is:
0016	basis incorrect - basic bit not set for:
0017	basis incorrect - duplicate index in basis list xh
0018	basis incorrect - index in xh out of range:
0019	insufficient main memory - increase xmreal to:
0020	insufficient main memory - increase xmreal
0021	insufficient main memory during presolve
0022	insufficient space for triplets - increase xnenmx
0023	Allocation of memory missing or unsuccessfull!
0024	MOPS library invalid!
0030	open error on file with unit:
0031	read error on file with unit:
0032	write error on file with unit:
0033	rewind error on file with unit:
0034	input file incorrect for your model - unit:
0035	error closing file with unit:
0036	license typ invalid for this feature:
0037	row limit exceeded - limit is:
0038	col limit exceeded - limit is:
0039	USB dongle not present or invalid
0040	imr is not written - set ximrrw = 1
0041	more then 3 primal-dual cycles
0050	name in profile unknown - line:
0051	apostrophe missing or character string too long
0054	double precision value in profile incorrect - line:
0055	integer value in profile incorrect - line:
0058	input record incomplete - missing value(s)
0060	token in record exceeds 64 characters

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

0061	invalid fc cost for sc/si or invalid value for pi
0062	lb of sc/si is less than zero
0063	si/sc,pi,sos or li are used: wrong first record
0110	premature end of input file
0112	name card missing in mps-data file
0113	rows section card missing in mps-data file
0122	no valid rows specified
0123	no objective function specified
0125	illegal row type:
0126	row name blank
0128	row name x at record y already present; row index of duplicate:
0129	hash table too small - increase xmxnen
0130	row limit exceeded - increase xmmax
0131	name space too small - increase xrcmax
0132	column section card missing
0152	column name missing
0156	nonzero limit to small - increase xnzmax or xadnon
0157	col name x at record y already present; col index of duplicate:
0158	column hash table too small - increase xmxnen
0159	col limit exceeded - increase xnmax
0161	duplicate row entry in column section
0162	row name unknown
0163	illegal numerical value in mps-file at record:
0164	nonzero different from 1.0 in sos row
0165	a non sos variable has a nonzero in a sos row
0166	wrong section card after column section
0167	magnitude of nonzero too large
0170	preceeding marker not correctly closed
0171	row name in sosorg marker card not found
0172	sos row was already defined
0173	sos row is not equality row
0180	rhs name missing
0181	too many nonzeros - increase xnzmax
0182	rhs value is different from 1.0 in sos row
0183	rhs value for an N-row is illegal
0184	xrhs appears twice
0185	wrong section card after rhs section
0186	row name x in rhs section at record y already present
0201	xrange appears twice
0202	wrong section card after range section

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

0203	duplicate row entry in range section
0222	xbound appears twice
0223	bound vector name missing
0224	illegal bound-type
0225	column name unknown
0226	bound redefined with different value or type
0227	problem infeasible due to non-integer bound
0228	wrong section card after bounds section
0301	first basis card is not a name card
0303	wrong type for basis card
0353	wrong type for basis card
0355	end marker e is missing in save basis file
0357	premature end of save basis file
0400	xm out of range
0401	xn is out of range; xn =
0402	xj is not equal to xm + xn
0403	xnzero out of range
0404	xjcp(xn + 1) - 1 is not equal to xnzero
0405	pointer xjcp is not increasing
0406	problem is unbounded
0407	duplicate row entry in column section
0408	row index out of range
0409	magnitude of element is smaller than xdropm
0410	magnitude of element is larger than xmaxel
0412	system error by checking logicals
0413	system error by checking bounds
0414	xrclen out of range
0415	inconsistency between xjcp(xn + 1) - 1 and xnzero
0419	column zero but xnenta not equal zero
0420	a fixed variable has an infinite bound - index is:
0421	lower bound larger than upper bound
0422	inactivated row in active part of column
0423	activate row in inactivated part of column
0424	xjcp(1) has to be positive
0425	xrcfre is greater than xrcmax
0426	xcoptr(1) has to be positive
0428	col index out of range
0429	name longer than xnamax characters
0430	xnenta negative
0431	sorting error detected in xcomp

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

0432	xptaij: xm = xn = 0
0500	function execution rejected; internal model is already transformed
0600	floating value is out of range - value is: x; range is from: y to z
0601	value is out of range - value is: x; range is from: y to z
0602	xsscal must be 1.d0 or -1.d0; its value is:
0650	input matrix is singular
0651	dense matrix is singular
0750	a fixed integer variable is nonintegral - index is:
0759	tree file incorrect, wrong tree or tree clobbered
0760	unresolvable num. problems in heuris at node:
0761	unresolvable num. problems in xipopt at node
0762	tree file empty - problem was already solved
0763	disk space too small to restart - increase xmxdsk!
0764	unknown node selection strategy
0800	a fixed bound is plus or minus xinf - xaprow
0801	row index out of range - xaprow
0802	row index not increasing - xaprow
0803	col index out of range - xaprow
0804	lower bound is larger than upper bound
0899	your test license expired
0900	fatal error in model generator
0901	copyright statement not correct

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

Chapter VII

MOPS Input Files

7 MOPS Input Files

7.1 MOPS Input Files

The MOPS Input Files pass data and parameter settings to MOPS.

- Profile
- Data Input

7.2 Profiles

All MOPS input parameters can be set in a Profile. Profiles are regular text files with the following structure:

- If a line starts with **, C* or *c* it contains a comment and will be ignored.
- After an exclamation mark (*!*) the rest of the line will be treated as a comment.
- All characters are converted to uppercase, if they are not part of a string enclosed by *"* or *'*. So MOPS parameter names are not case sensitive; *xStart* and *XSTARt* are the same.
- Values are assigned to MOPS parameters by the equal sign "=". Between parameter name, equal sign and value blanks can be set optionally.

Profiles can be read by the DLL function *ReadProfile.*

Example of a MOPS Profile:

```
* Filename could be 'myProfile.pro'
xfnmps = 'opti\probl\mops.mps'
xminmx = 'max' ! optimization direction min or max (default: 'min')
xstart= 1 ! start from crash basis
Xfnsta = 'statist' ! file name for statistic file
xmreal = 100 xnzmax = 50000 xadnon = 40000 xstorn = 0
```

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

7.3 Data Input

7.3.1 Data Input

The data input files pass model data to MOPS. The new branch and bound supports extended Integer Types, which need a special handling in the MPS and Triplet format.

- MPS Files
- Triplet Files
- Extended Integer Types

7.3.2 MPS Files

7.3.2.1 MPS Files

The MPS file format is can be processed by almost all commercial solver systems. Therefore MPS models are the ideal way of interchanging models between different MP systems. Note, that MOPS supports only the original MPS-format from IBM where the data is positioned in fixed fields.

The general structure of arranging date within an MPS file is as follows:

```
NAME     Name_of_model
ROWS
   Here constraints and objective function are specified.
COLUMNS
   Here variables and nonzero elements are specified column-wise.
RHS
   Here the right hand sides are listed.
RANGES
   Here individual bound for constraints can be specified.
BOUNDS
   Here individual bounds for single variables can be specified.
ENDDATA
```

The following main rules apply to any MPS file:

- A data deck starts always with the token NAME and ends with the ENDATA keyword.
- A data deck contains different sections, which start with the keywords: NAME, ROWS, COLUMNS, RHS, RANGES and BOUNDS.

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

- These sections must always be there: NAME, ROWS and COLUMNS. Other sections are optional.
- Within a section, data is stored row-wise and a row may not be longer than 80 characters.
- One data deck can contain several RHS, RANGES and BOUNDS sections.
- Comments are have an asterisk (*) in column 1 and will be ignored by the processor.

A data set (=one row of an MPS file) consists of six data fields:

Field	1	2	3	4	5	6
Columns	2-3	5-12	15-22	25-36	40-47	50-61
Content	Type	Name1	Name2	Value1	Name3	Value2

Furthermore the following rules must be considered:

- An MPS data deck should not contain more than one model. Several N-constraints, RHS, RANGES and BOUNDS vectors are possible.
- Names are fixed length strings of 8 characters. Two names with different numbers of blanks at the beginning and end are regarded as different. Example: The names "bbJACKbb" and "bbbJACKb" are different with "b" representing a blank character.
- Row and column names are separately saved. Rows and columns can have the same name, but it is not recommended.
- MOPS does not accept "Dx-Rows" in the ROWS section, i.e. a row may not be a linear combination of other constraints. Scaling factors identified by the keyword SCALE in field 2 and a scaling value in field 4 will be ignored.
- Also scaling information will not be supported in the COLUMNS section.
- Integer variables must have upper and lower bound in the intervall [-32767, 32767]. If integer variables exceed these values, the respective bound will be set accordingly without an error message or a warning.

7.3.2.2 Example

Example of an IP model

Min X_4

R1: $0.5\ X_1 - X_2 + X_3 + X_4 = 1$

R2: $0.5\ X_1 - X_2 \quad - X_4 = 0$

$X_1 \leq -2$

$X_2 \geq 0$

$X_3 \leq 20$

$X_4 \leq 30$ and integer

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

```
1    5               15              25              40              50
NAME                 NONAME

ROWS
 N   OBJ-FUNC
 E   R1
 E   R2
COLUMNS
     X1              R2              0.50000         R1              0.50000
     X2              R2              -1.0000         R1              -1.0000
     X3              R1               1.0000
     X4              OBJ-FUNC         1.0000         R2              -1.0000
     X4              R1               1.0000
RHS
     RHS             R1               1.0000
BOUNDS
 UP  BOUND           X1              -2.00
 UP  BOUND           X3              20.00
 UI  BOUND           X4              30.00
ENDDATA
```

The first line is not part of the MPS file, it is only to show column positions.

Another example for an MPS file can be found in the Burma case study.

7.3.2.3 Sections

NAME Section

The NAME section contains the keyword NAME in column 1. Starting from column 15 the name of the model is specified as a string, that may not contain blanks. Only the first 32 characters of this string are significant.

ROWS Section

The ROWS section starts with a line that contains the keyword ROWS in column 1. In the following rows the constraints of the model are specified. Field 1 contains the type of the constraint: L or LE for "less or equal", G or GE for "greater or equal", E or EQ for "equal" and N for "unrestricted". The objective function is always specified as N. Types must be in capital letters.

In field 2 follows the name of the constraint as a 8 character string, with leading and trailing blanks being significant.

COLUMNS Section

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

The COLUMNS section starts with the keyword COLUMNS in column 1. The following rows contain variable names and nonzero elements of the matrix. A nonzero element is specified by the respective row name (same as in ROWS section) and the column name. Again leading and trailing blanks are significant. Field 2 contains the column name, field 3 holds the row name and field 4 stores the value of the nonzero element. Values must be in US number format (e.g. -1000.1). Exponents are denoted by "D" or "E" and optionally a sign. Examples of valid numbers are: *1.1E-2, 27, 0.27, + 0.27, -2, -27.E+5.*

Fields 5 and 6 can optionally contain another nonzero element of the same column. All nonzero elements of a column must be grouped together; their order within this group is irrelevant.

RHS Section

The RHS section starts with keyword RHS in column 1. In the following datasets the nonzero elements of the right hand side vector are specified. several vectors are possible, but usually only one vector is specified. Each vector must have an unique name in field 2. In fields 3 to 6 the nonzero elements are stored like in the COLUMNS section. The same rules like in the COLUMNS section apply.

RANGES Section

The RANGES section starts with keyword RANGES in column 1. The following datasets hold the nonzero elements of the RANGES vector. Each vector must have an unique name in field 2. In fields 3 to 6 the nonzero elements are stored like in the COLUMNS section. The same rules like in the COLUMNS section apply. A RANGES vector defines upper and lower bound of a constraint:

The bounded constraint $l <= a\ x <= u$, with a and x being real vectors and l and u being real scalars, can be defined by a RANGES vector. In the ROWS section this constraint can be defined as GE or LE hence in the RHS section it will hold either l or u (if not 0) as its coefficient. Be $r = u-l$, with $r > 0$. Then the RANGES section contains for this constraint r as coefficient. A RANGES element can also be used to specify an equation.

Generally speaking, if b is the coefficient of a constraint defined in the RHS section, r is the coefficient specified in the RANGES section and if l and u are the lower and upper bound of the constraint, the following relationship applies:

Type	Sign of r	l	u
G	+ or -	b	b+ |r|
L	+ or -	b - |r|	b
E	+	b	b+|r|
E	-	b-|r|	b

BOUNDS Section

The BOUNDS section starts with keyword BOUNDS in column 1. In the following datasets individual bounds for the structural variables can be defined, if these bound differ from the default bounds. Default is 0 as lower bound and infinite as upper bound. Each vector must have an unique name in field 2. Field 3 contains the name of the structural variable and field 1 holds the type of the variable. Some types require field 4 to hold a numeric value for the respective bound. The following types are possible:

Type	Meaning / Intervall	Value required (Y/N)
LO	Lower Bound	Y
UP	Upper Bound	Y
FX	Fixed Value	Y

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

FR	Free Variable	N
PL	[0,∞)	N
MI	(∞, 0]	N
BV	Binary Variable	N
LI	Integer Variable Lower Bound	Y
UI	Integer Variable Upper Bound	Y

Sometimes more than one line in a BOUNDS section is necessary to specify bounds:

-20 <= x <= 30 requires *LO x -20* and *UP x 30*

-inf < x <= 10 requires *MI x* und *UP x 10*

7.3.2.4 Integer Variables

The normal integer variables (binary, general integer and SOS Type 3) can be declared in MPS files in two ways:

1. By enclosing the integer variables in MPS Marker Datasets. A Marker Dataset contains blanks in field 1, a specific name in field 2 and field 3 contains the token 'MARKER' (with inverted comma). Field 4 is empty and field 5 contains a one of the following keywords: 'INTORG', 'INTEND', 'SOSORG', or 'SOSEND'.

   All variables enclosed in Marker Datasets with the keywords 'INTORG' and 'INTEND' are integer. If no further specification about these variables are made in the BOUNDS section, they will be regarded as 0/1 variables. For defining them as integers, appropriate bounds must be set in the BOUNDS section.

   Variables enclosed between the keywords 'SOSORG' and 'SOSEND' belong to a Special Ordered Set (SOS). MOPS will generate for SOS variables a constraint, in which all variables of the set have the coefficient 1.0 and the value of the RHS is also 1.0, with all variables of the set being binary variables. So in a feasible solution one of the SOS variables must have the value one, all others must be zero.

```
...
ROWS
 E   SON1102
COLUMNS
     SON1102          'MARKER'                         'SOSORG'
     Y2N1101          KSN1101          1.00000
     Y2N1101          RTN1101         -5.00000
     Y2N1102          RTN1102          5.00000
     Y2N1103          RTN1103          5.00000
     Y2N1104          KSN1104          1.00000
     Y2N1105          RTN1105          2.00000
     Y2N1106          KSN1106         -1.00000
     SEN1999          'MARKER'                         'SOSEND'
...
```

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

```
     IONW999           'MARKER'                          'INTORG'
     XXNW100           GGNW100            -2500.00000
     XXNW100           GONW100           -90000.00000
     IENW999           'MARKER'                          'INTEND'
...
```

The names, that precede in field 2 the 'MARKER' tokens are irrelevant for MOPS.

2. Integer variables can also be specified in the BOUNDS section by the respective type keywords: BV defines a binary variable and LI/LU define lower and upper bound of a variable, that becomes implicitly integer.

7.3.3 Triplet Files

7.3.3.1 Triplet Format

Triplet files are a MOPS-specific file format for LP and IP models. The Triplet file format has several advantages against MPS files:

- Fast to read
- Free of redundancies
- Names can be longer than 8 characters (up to 64)

The five Sections of the Triplet Format

Section 1 contains comments. All lines at the beginning of a Triplet file, that start with * are treated as comments, they are read over by the MOPS parser.

Section 2 holds model dimensions. This section consists of only one line, which holds the following information in the given order:

- Number of constraints (xm)
- Number of variables (xn)
- Max Number of nonzeros (xnzmax)
- Max number of 1-byte characters used for variable and constraints names (xrcmax)
- Floating point value to represent infinity (e.g. 1.0E+20)

The first four values are integer. If *xrcmax = 0*, no names will be processed, even if there are names present in the Triplet file. If *xrcmax = 1*, then xrcmax will be calculated as *xrcmax = (xn + xm) * xdfnal,* where *xdfnal* specifies the maximum length of names (default: xdfnal =*8)*. If *xrcmax > 1*, its actual value will be taken.

Section 3 contains *xn* datasets with information for all structural variables. Each line holds the following values in the given order:

- Lower bound
- Upper bound
- Cost coefficient

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

- Type (0: continuous, 1: integer)
- Name of variable (optionally, max 64 characters). The name may be enclosed in quotation marks ("). Inverted commas or quotation marks are treated as delimiters and will not be stored with the name.

Section 4 contains *xm* datasets with information for all constraints. Each line holds the following values in the given order:

- Cost coefficient.
- MOPS internal lower bound (internal LHS).
- MOPS internal upper bound (internal RHS).
- Name of constraint (optionally, max 64 characters). The name may be enclosed in quotation marks ("). Inverted commas or quotation marks are treated as delimiters and will not be stored with the name.

The internal bounds are defined as:

MOPS internal lower bound = - external upper bound

MOPS internal upper bound = - external lower bound

Section 5 holds the nonzero elements of the matrix. Each nonzero element is stored in a separate line as Triplet (i,j,aij) of row index i, column index j and nonzero value aij. The indices must be in concordance with the numbering of rows and columns used in section 3 and 4. i and j are integers with $1 \leq i \leq xm$ and $1 \leq j \leq xn$.

Further format specifications

- Comments are only allowed at the beginning of a Triplet file.
- Data items (names and values) are treated as tokens, which are divided by the separator characters blank, comma and semicolon. These characters may not appear in tokens.
- Integers may not contain a decimal point, even if the resulting value is integer (123.0 is not allowed). Floating point values must contain a decimal point.
- Names from a Triplet file will only be stored in MOPS if *xstorn = 1*, which is the default for xstorn. If xrcmax = *0*, *xstorn* will be set to 0 and no names will be stored.

Error codes

If an error occurs when reading a Triplet file, the return code from the DLL function *ReadTripletFile()* will be 2 and you can check MOPS parameter xertyp with DLL function *GetParameter()* for further information. If you are working with the static MOPS.LIB, check xrtcod and xertyp directly. See *ReadTripletFile()* for information about possible error types.

7.3.3.2 Example

Example of an IP model

Min X_4

R1: $0.5\ X_1 - X_2 + X_3 + X_4 = 1$

R2: $0.5\ X_1 - X_2 \quad - X_4 = 0$

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

$X_1 \leq -2$

$X_2 \geq 0$

$X_3 \leq 20$

$X_4 \leq 30$ and integer

```
************************************
*
* example of an IP model
************************************
*
2,4,7,221,1E+20
-1E+20,-2,0,0,'X1'
0,1E+20,0,0,'X2'
-20,20,0,0,'X3'
-1E+20,1E+20,1,1,'X4'
-1,-1,'R1'
0,0,'R2'
1,1,0.5
1,2,-1
1,3,1
1,4,1
2,1,0.5
2,2,-1
2,4,-1
```

Another example for an Triplet file can be found in the Burma case study.

7.3.4 Extended Integer Types

7.3.4.1 Extended Integer Types

The new Branch and Bound (xnewbb=1) in MOPS supports the following integer types:

Name	Type Number
binary variables for linearized functions (LI-variable)	<0
continuous variables	0
binary variables	1
integer variables	2
semi-continuous variables (SC-variable)	3
semi-integer variables (SI-variable)	4
partial-integer variables (PI-variable)	5
SOS of type 3	> 5 and <= 5 + xm

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

SOS of type 1	> 5 + xm and <= 5 + 2*xm
SOS of type 2	> 5 + 2*xm

All variables of the same SOS or the same linearized function must have the same ***unique*** type number.

The extended integer types can require additional information:

Variable	Information	Required
semi-continuous variables	- fix charge cost - threshold value	- no - yes
semi-integer variables	- fix charge cost - threshold value	- no - yes
partial-integer variables	- threshold value	- yes
SOS of type 1,2,3	- weights	- no

The threshold values of the semi-continuous and semi-integer variables and the weights of the SOS are saved in the lower bounds of the belonging variables. .

There are three possible ways to define the weights for a SOS:

- Reference row: only the first element of this set will save the row index of the reference row in the lower bound. The remaining lower bounds are set to zero.
- Weights: they are saved for all SOS-variables in the lower bound.
- No specification: the lower bounds of the SOS variable are zero. The weights are the value of the decreasing order.

To improve the performance, the weights have to be in a decreasing order.

The extended integer types need a enhanced MPS and Triplet Format.

7.3.4.2 Enhanced MPS Format

Integer variables with extended integer types need an enhanced MPS format, which provides additional marker in the column section and additional types in the bound section.

Column section

In this section SOS variables and binary variables for linearized functions have to be marked.

SOS
The SOS variables are marked with the keywords SOSORG and SOSEND (see Integer Variables in MPS). To define the three types of Special Ordered Sets, the first field of the record include a ‘S1’ for the first, a ‘S2’ for the second and blanks for the third type of SOS.

```
...
ROWS
 E    SOS3
 L    SOS1
 L    SOS2
COLUMNS
```

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

```
     SOS3                  'MARKER'                      'SOSORG'
     ...                   ...              ...
     SOS3END               'MARKER'                      'SOSEND'

 S1  SOS1                  'MARKER'                      'SOSORG'
     ...                   ...              ...
     SOS1END               'MARKER'                      'SOSEND'

 S2  SOS2                  'MARKER'                      'SOSORG'
     ...                   ...               ...
     SOS2END               'MARKER'                      'SOSEND'
...
```

When a variable and its type are defined as a SOS variable with type x, then it is not necessary to define the nonzero elements of the SOS-row and there bounds in the bound section (exception: weights in the lower bound, see Extended Integer Types).

Linearized functions
The LI-variables of linearized functions are integer variables and each group of LI-variables are marked with the keywords INTORG and INTEND. To identify the groups, they are indicated with a 'LI' in the first field of the record.

```
COLUMNS
 LI  LI1                   'MARKER'                      'INTORG'
     ...                   ...              ...
     LI1END                'MARKER'                      'INTEND'
```

These variables are saved as binary variables and so it is not necessary to define their bounds in the bound section.

Bound section

The extended integer types require new types in the bound section:

Type	Meaning	Value required (Y/N)
SC	fix charge costs for SC-variables	N
SI	fix charge costs for SI-variables	N
PI	threshold value for PI-variables	Y

Semi-continuous variable
This variable is marked with the type SC. The threshold value and the upper bound are labeled with the types LO and UP. If no threshold value is defined or it is set to zero, then the threshold value is set to xtolin. If the upper bound is not defined it is set to xinf. The lower bound has to be greater or equal zero and the upper bound has to be greater than the threshold value. A special order of the types LO, UP and SC is not necessary.

```
BOUNDS
```

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

```
LO  BND          SC1          2.000
UP  BND          SC1          9.000
SC  BND          SC1          20.00
LO  BND          SC2          2.000
UP  BND          SC2          9.000
SC  BND          SC2
LO  BND          SC3          5.000
SC  BND          SC3
```

The SC-Variable SC2 and SC3 have no fix charge costs. The upper Bound of variable SC3 is xinf.

Semi-integer variable
This variable is marked with the type SI. The threshold value and the upper bound are labeled with the types LO and UP or with LI and UI. If LO/LI is set to zero or is not defined, then the threshold value is set to one. If the upper bound is not defined it is set to xinf. The threshold value has to be greater or equal zero and the upper bound has to be greater than the threshold value. A special order of the types LO/LI, UP/UI and SI is not necessary.

```
BOUNDS
 LO  BND          SI1          2.000
 UP  BND          SI1          9.000
 SC  BND          SI1          20.00
 LI  BND          SI2          2.000
 UI  BND          SI2          9.000
 SC  BND          SI2
 UP  BND          SI3          20.00
 SC  BND          SI3
```

The SI-Variable SI2 and SI3 have no fix charge costs. The threshold value of variable SI3 is one.

Partial integer variable
This variable is marked with the type PI. The lower and upper bound are labeled with the types LO/LI and UP/UI. The threshold value of this variable has to be greater than the lower bound and less than the upper bound.

```
BOUNDS
 LO  BND          PI1          2.000
 UP  BND          PI1          40.00
 PI  BND          PI1          20.00
 LI  BND          PI2          4.000
 UI  BND          PI2          9.000
 PI  BND          PI2          6.000
```

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

7.3.4.3 Enhanced Triplet Format

The integer variables with a extended integer type needs an enhanced Triplet format

The declaration of the model dimension in **Section 2** gets an additional parameter. If the value of this parameter is greater than zero, the model contains variables with extended variable types.

The **Section 3** has to be adapted to the new variable types. In this section the variable is defined with its lower bound, upper bound, cost, type and optional with its name. The changes of the lower bound and the different variable types are already described. The definition of the fix charge costs of SC- and SI-variables and the threshold value of the PI-variable need a new record. The record follows the normal definition of the variable. It is marked with a ‘k’ or ‘K’ at the beginning and includes the value for the fix charge costs or the threshold value.

```
***********************************
*
* example with extended integer
types
***********************************
*
100,200,500,221,1E+20,1
0,1,0,-1,'LIVar1'
0,1,0,-1,'LIVar2'
0,1,0,-1,'LIVar3'
4,1,0,406,'SOS2Var1'
0,1,0,406,'SOS2Var2'
0,1,0,406,'SOS2Var3'
1,10,-20,3,'SC'
k,30
2,20,50,4,'SI'
k,10
0,1000,7,5,'PI'
K,300
...
```

The variables LIVar1-LIVar3 belong to a linearized function. The integer type has to be less than zero. The variables SOS2Var1-SOS2Var3 are from the Special Ordered Set Type 2. The weights of these variables are included in the reference row number four. The integer type has to be greater than 2*xm+5. The SC-variable has the threshold value 1 and fixed charge cost 30. The SI-variable has the threshold value 2 and the fix charge cost are 10. The threshold value of the PI-variable is 300.

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

Chapter VIII

MOPS Output Files

8 MOPS Output Files

8.1 MOPS Output Files

MOPS does a considerable part of output via files. The files written by MOPS are plain text files, which can be edited with almost any text editor. See the following sections for more information about MOPS files:

- Solution Files
- Log Files
- fort.xxx Files

8.2 Solution Files

If parameter *xoutsl* has been set accordingly, the LP and -if existing- an IP solution will be written to files *xfnlps* and *xfnips*. Solution files consist of tree sections:

1. Identification

Gives information about status (optimal, infeasible, unbounded) and the objective function value.

2. Row Section

For each row the following information is given:

- Internal index (*xn + 1, ... xn + xm).*
- Status:
  BS Variable is in basis.
  UL Variable is at upper limit.
  LL Variable is at lower limit.
  EQ Variable ist fixed (UL = LL).
  IN Restriktion is inactive.
- Row name. If the name is longer than 8 characters, only the leading 8 characters will be written.
- Activity: Value of the constraint when solution values are put in.
- Slack: Difference between the value of the constraint and the RHS bound.
- Lower limit (LHS).
- Upper limit (RHS).
- Dual Activity: Indicates the marginal change of the objective function value, if the right hand side would be changed with 1 unit.

3. Column section

- Internal index (*1, ..., xn).*
- Status like in Row Section. In IP models, integer variables have status IV.
- Column name. If the name is longer than 8 characters, only the leading 8 characters will be written.
- Activity: Solution values of variables.
- Objective function coefficient.
- Lower limit of variable.

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

- Upper limit of variable.
- Reduced Cost: Indicates that the objective function will change with the value of the reduced cost if the constraint is changed with 1 unit.

Note

The preprocessor (*xrduce = 1*) might change the bounds of structural variables, but this does of course not change the LP solution. If you want the bounds to remain unchanged, keep the default *xbndha = 0.*

8.3 Log Files

If parameter *xoutlv* >= *2*, a log file will be opend and all major MOPS routines will write a log message to that file when they are called.

The name of the log file can be specified by parameter *xfnlog*. If no name has been set, the log file's name is *for007* or *fort.7.*

Logging, especially with *xoutlv* >= *3,* is very time-consuming and should only be activated for debugging purposes during application development.

8.4 fort.xxx Files

MOPS creates a number of output files in the working directory during an optimization run. If you don't specify file names explicitly, the names of these files have formats like *forxxx or fort.xxx*, with *xxx* being an internal Fortran unit number. The following table shows which parameters you have to set to give these files explicit names and which parameters impede the creation of the files. When using MOPS.DLL, call *SetParameter* to make the appropriate settings.

File name	File type	Own file name	Don't create
fort.7	Log file	*xfnlog = myName.lps*	*xoutlv = 1*
fort.13	LP solution file	*xfnlps = myName.lps*	*xoutsl = 0*
fort.14	External basis file	*xfnbao = myName.bao*	*xfrbas = -1*
fort.16	IP solution file	*xfnips = myName.ips*	*xoutsl = 0*
fort.17	Internal basis file	*xfnsav = myName.sav*	*xfrbas = -1*
fort.19	Statistic file	*xfnsta = myName.sta*	*xoutlv = 0*
fort.20	Error message file	*xfnmsg = myName.msg*	*xoutlv = 0*
fort.24	Tree	*xfntre= myName.tre*	*xfrtre = -1*

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

See also

xoutlv, xoutsl, xfrbas, xfrtre

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

Chapter IX

Case Burma

9 Case Burma

9.1 Problem

The government of Burma plans the agricultural development of a 20 square kilometers (sqkm) region, that can be divided into 4 parts, each 5 sqkm large. In each of these parts, rice or wheat can be cultivated. We have to decide how many sqkm rice and how many sqkm wheat to cultivate. The most important factor is the construction of irrigation facilities. The following investments in irrigation facilities are possible (projects P1 to P5):

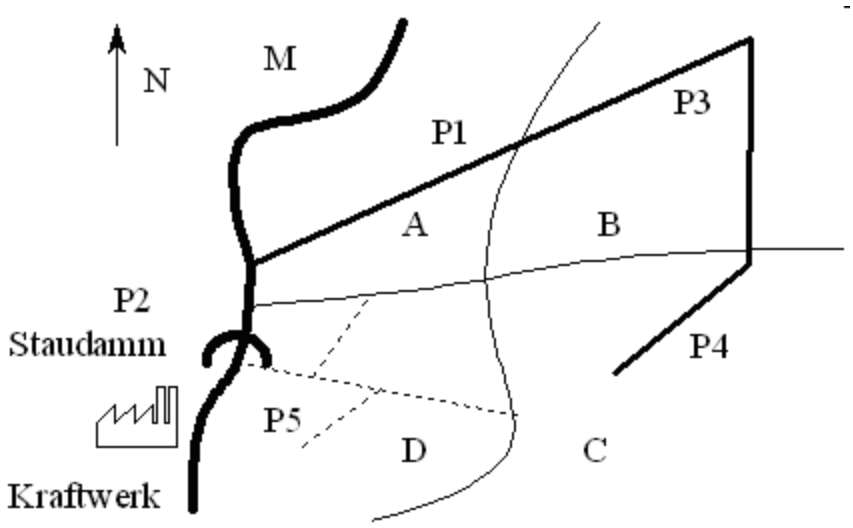
P1: Area A can be irrigated by a canal from the river M (see sketch: river M is flowing from north to south).

P2: Building an embankment dam, so other regions can also be irrigated.

P3: If projects P1 and P2 are realized, the canal can be extended to bring water to area B.

P4: Area C can be irrigated to another channel, starting from the P3 channel.

P5: Additionally to the embankment dam, but independently from P1, P3 and P4 a hydro power plant can be constructed to irrigate region D with electrical pumps. This project will not have any effect on the other areas.



The government intends to realize at least one of the projects P1 to P5. The cultivation of rice and wheat differs in the consumption of water and in yield. (see data below). Because of certain market constraints the area to cultivate wheat on may not exceed 30% of the total area used for agriculture. The planning period are 12 years and 2 years are needed to realize a project. We are looking for the solution with maximal profit.

Data

Amount of water in (1000 cubic meter):

In area A 1.5, if P1 is realized.

In area A 1.4 and in area B 1.0, if P3 is realized.

In area A 1.0, in area B 0.8 and in area C 0.6, if P4 is realized.

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

In area D 1.2, if P5 is realized.

Investment costs (in 1000 $):

P1: 5000; P2: 10000; P3: 2000, P4: 2000 ; P5: 5000.

Water consumption (in 1000 cubic meters / sqkm):

Rice: 0.25

Wheat: 0.1

Profit (in 1000 $/sqkm):

Area A: Rice 1100. Wheat 1000.

Area B: Rice 1300. Wheat 1300.

Area C: Rice 1400. Wheat 1600.

Area D: Rice 1600. Wheat 1800.

Task

Build a linear mixed integer problem to maximize profit.

9.2 Solution

Decision variables

xga: Area under cultivation for wheat in region A (in sqkm)

xgb: Area under cultivation for wheat in region B (in sqkm)

xgc: Area under cultivation for wheat in region C (in sqkm)

xgd: Area under cultivation for wheat in region D (in sqkm)

xra: Area under cultivation for rice in region A (in sqkm)

xrb: Area under cultivation for rice in region B (in sqkm)

xrc: Area under cultivation for rice in region C (in sqkm)

xrd: Area under cultivation for rice in region D (in sqkm)

y1 = 1 if area A is irrigated by a channel, else y1 = 0

y2 = 1 if the embankment dam is build, else y2 = 0

y3 = 1 if the channel to irrigate area B is build , else y3 = 0

y4 = 1 if the channel to irrigate area C is build, else y4 = 0

y5 = 1 if the hydro plant and pumping irrigation for region D is build, else y5 = 0

Objective function

Maximize z, with

z = 1100 xra + 1300 xrb + 1400 xrc + 1600 xrd + 1000 xga + 1300 xgb + 1600 xgc + 1800 xgd - 5000 y1 - 10000 y2 - 2000 y3 - 2000 y4 - 5000 y5

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

constraints

a. Logical constraints

P3 may only executed if both P1 and P2 are executed:

$2\ y3 \leq y1 + y2$ (or better as disaggregated formulation: $y3 \leq y1$, $y3 \leq y2$)

P4 may only be executed if P3 is executed:

$y4 \leq y3$

P5 may monly be executed if the embankment dam is build:

$y5 \leq y2$

At least one of the projects must be accomplished:

$y1 + y2 + y3 + y4 + y5 \geq 1$

b. Irrigation constraint

Region A: $1.5\ y1 - 0.1\ y3 - 0.4\ y4 \geq 0.25\ xra + 0.1\ xga$

Region B: $1.0\ y3 - 0.2\ y4 \geq 0.25\ xrb + 0.1\ xgb$

Region C: $0.6\ y4 \geq 0.25\ xrc + 0.1\ xgc$

Region D: $1.2\ y5 \geq 0.25\ xrd + 0.1\ xgd$

c. constraint of the area under cultivation

$xra + xga \leq 5,\ xrb + xgb \leq 5,\ xrc + xgc \leq 5,\ xrd + xgd \leq 5$

d. Maximal wheat cultivation

$xga + xgb + xgc + xgd \leq 0.3\ (xga + xra + xgb + xrb + xgc + xrc + xgd + xrd)$

Types of variables

y1, y2, y3, y4, y5 are binary variables.

xga, xgb, xgc, xgd, xra, xrb, xrc, xrd are nonnegative continues variables.

Tableau representation of the model

	xra	xrb	xrc	xrd	xga	xgb	xgc	xgd	y1	y2	y3	y4	y5	Typ	RHS
profit	1100	1300	1400	1600	1000	1300	1600	1800	-5000	-10000	-2000	-2000	-5000		
project3									-1	-1	2			<=	0

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

project4											-1	1		<=	0
project5										-1			1	<=	0
p1to5									1	1	1	1	1	>=	1
waterA	0.25				0.1				-1.5		0.1	0.4		<=	0
waterB		0.25				0.1					-1	0.2		<=	0
waterC			0.25				0.1					-0.6		<=	0
waterD				0.25				0.1					-1.2	<=	0
areaA	1				1									<=	5
areaB		1				1								<=	5
areaC			1				1							<=	5
areaD				1				1						<=	5
wmax30	-0.3	-0.3	-0.3	-0.3	0.7	0.7	0.7	0.7						<=	0
LB	0	0	0	0	0	0	0	0	0	0	0	0	0		
UB									1	1	1	1	1		
Typ	cont	cont	cont	cont	cont	cont	cont	cont	bin	bin	bin	bin	bin		

9.3 MPS File

MPS file for the "Burma" model

```
NAME          BURMAA
ROWS
 N  OBJECTIV
 L  LOGISCH1
 L  LOGISCH3
 L  LOGISCH4
 G  LOGISCH5
 L  WASSERA
 L  WASSERB
 L  WASSERC
 L  WASSERD
 L  ANBAUA
 L  ANBAUB
 L  ANBAUC
 L  ANBAUD
 L  MXGETREI
COLUMNS
    REISA     OBJECTIV      1100.000   WASSERA           .250
```

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

```
    REISA     ANBAUA          1.000   MXGETREI         -.300
    REISB     OBJECTIV     1300.000   WASSERB           .250
    REISB     ANBAUB          1.000   MXGETREI         -.300
    REISC     OBJECTIV     1400.000   WASSERC           .250
    REISC     ANBAUC          1.000   MXGETREI         -.300
    REISD     OBJECTIV     1600.000   WASSERD           .250
    REISD     ANBAUD          1.000   MXGETREI         -.300
    GETREIDA  OBJECTIV     1000.000   WASSERA           .100
    GETREIDA  ANBAUA          1.000   MXGETREI          .700
    GETREIDB  OBJECTIV     1300.000   WASSERB           .100
    GETREIDB  ANBAUB          1.000   MXGETREI          .700
    GETREIDC  OBJECTIV     1600.000   WASSERC           .100
    GETREIDC  ANBAUC          1.000   MXGETREI          .700
    GETREIDD  OBJECTIV     1800.000   WASSERD           .100
    GETREIDD  ANBAUD          1.000   MXGETREI          .700
    IST00001  'MARKER'                'INTORG'
    EIKANALA  OBJECTIV    -5000.000   WASSERA         -1.500
    EIKANALA  LOGISCH1       -1.000   LOGISCH5         1.000
    STAUDAMM  OBJECTIV   -10000.000   LOGISCH1        -1.000
    STAUDAMM  LOGISCH5        1.000   LOGISCH4        -1.000
    ANKANALB  OBJECTIV    -2000.000   WASSERA           .100
    ANKANALB  WASSERB        -1.000   LOGISCH1         2.000
    ANKANALB  LOGISCH3       -1.000   LOGISCH5         1.000
    ANKANALC  OBJECTIV    -2000.000   WASSERA           .400
    ANKANALC  WASSERB          .200   WASSERC          -.600
    ANKANALC  LOGISCH3        1.000   LOGISCH5         1.000
    KRAFTWER  OBJECTIV    -5000.000   WASSERD         -1.200
    KRAFTWER  LOGISCH4        1.000   LOGISCH5         1.000
    IEN00001  'MARKER'                'INTEND'
RHS
    RHS       ANBAUA          5.000   ANBAUB           5.000
    RHS       ANBAUC          5.000   ANBAUD           5.000
    RHS       LOGISCH5        1.000
BOUNDS
 UP BOUNDS    EIKANALA        1.000
 UP BOUNDS    STAUDAMM        1.000
 UP BOUNDS    ANKANALB        1.000
 UP BOUNDS    ANKANALC        1.000
 UP BOUNDS    KRAFTWER        1.000
ENDATA
```

9.4 Triplet File

Triplet file for the "Burma" model

```
13 13 47 202 1e+020
0 1e+020 1100 0 'REISA'
0 1e+020 1300 0 'REISB'
0 1e+020 1400 0 'REISC'
0 1e+020 1600 0 'REISD'
0 1e+020 1000 0 'GETREIDA'
0 1e+020 1300 0 'GETREIDB'
0 1e+020 1600 0 'GETREIDC'
0 1e+020 1800 0 'GETREIDD'
0 1 -5000 2 'EIKANALA'
0 1 -10000 2 'STAUDAMM'
0 1 -2000 2 'ANKANALB'
0 1 -2000 2 'ANKANALC'
0 1 -5000 2 'KRAFTWER'
```

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

```
0 1e+020 'LOGISCH1'
0 1e+020 'LOGISCH3'
0 1e+020 'LOGISCH4'
-1e+020 -1 'LOGISCH5'
0 1e+020 'WASSERA'
0 1e+020 'WASSERB'
0 1e+020 'WASSERC'
0 1e+020 'WASSERD'
-5 1e+020 'ANBAUA'
-5 1e+020 'ANBAUB'
-5 1e+020 'ANBAUC'
-5 1e+020 'ANBAUD'
0 1e+020 'MXGETREI'
1 9 -1
1 10 -1
1 11 2
2 11 -1
2 12 1
3 10 -1
3 13 1
4 9 1
4 10 1
4 11 1
4 12 1
4 13 1
5 1 0.25
5 5 0.1
5 9 -1.5
5 11 0.1
5 12 0.4
6 2 0.25
6 6 0.1
6 11 -1
6 12 0.2
7 3 0.25
7 7 0.1
7 12 -0.6
8 4 0.25
8 8 0.1
8 13 -1.2
9 1 1
9 5 1
10 2 1
10 6 1
11 3 1
11 7 1
12 4 1
12 8 1
13 1 -0.3
13 2 -0.3
13 3 -0.3
13 4 -0.3
13 5 0.7
13 6 0.7
13 7 0.7
13 8 0.7
```

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

9.5 AMPL

AMPL files for the "Burma" model

Model file

```
#sets
set REGION; # the four parts of the 20 square kilometer region
set PROD;   # wheat and rice
set PROJ;   # the five possible projects

#parameter
param water{REGION,PROJ} default 0.0; #the amount of water
param costs{PROJ} >= 0; # the investment costs
param wateruse{PROD} >= 0.0; # the water consumption
param profit{REGION,PROD} >= 0; # profit
param maxarea{REGION} default 5; # max. size of each area
param percent default 0.3; # percent of the whole region to cultivate wheat

#variables
var project{PROJ} binary;
var area{PROD,REGION} >= 0;

# objective function
maximize ProfitBurma: sum{r in REGION,p in PROD} profit[r,p]*area[p,r] -
         sum{j in PROJ}costs[j]*project[j];

# Logical constraints
s.t. Project3: 2*project['P3'] - project['P1'] - project['P2'] <=0;
s.t. Project4: project['P4'] - project['P3'] <= 0;
s.t. Project5: project['P5'] - project['P2'] <= 0;
s.t. AllProjects: sum{j in PROJ} project[j] >= 1;

# Irrigation constraint
s.t. Waterbalance {r in REGION}: sum{j in PROJ} water[r,j]*project[j] >=
     sum{p in PROD} wateruse[p]*area[p,r];

# constraint of the area under cultivation
s.t. Maximal_Area {r in REGION}: sum{p in PROD} area[p,r] <= maxarea[r];

# maximal wheat cultivation
s.t. Maximal_Wheat: sum{r in REGION} area['W',r] - sum{p in PROD,r in
REGION}
     percent*area[p,r] <= 0;
```

Data file

```
#sets
set REGION := A B C D;
set PROD := W R;
set PROJ := P1 P2 P3 P4 P5;

#parameter
param water:=
: P1   P3   P4   P5 :=
A 1.5 -0.1 -0.4   .
B  .   1.0 -0.2   .
C  .    .   0.6   .
```

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

```
D  .    .    .   1.2;

param costs := P1 5000 P2 10000 P3 2000 P4 2000 P5 5000;

param wateruse := W 0.1 R 0.25;

param profit:=
:  W     R  :=
A  1000   1100
B  1300   1300
C  1600   1400
D  1800   1600;
```

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

Chapter

References

10 References

10.1 References

MOPS White Paper: **M**athematical **OP**timization **S**ystem, Technical Description, MOPS Optimierungssysteme GmbH & Co. KG, Paderborn, January 2007, available as download from the MOPS homepage mops-optimizer.com

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

Index

- 6 -

64-Bit 49

- A -

AllocateMemory 8, 51

- B -

Burma 142

- C -

C# 47
Case 142
ChangeColLB 22
ChangeColType 22
ChangeColUB 22
ChangeCost 22
ChangeLhs 22
ChangeNonzeros 22
ChangeRhs 22
C-Sharp 47

- D -

DelCol 22, 51
DelRow 22, 51

- F -

FindName 22, 51
Finish 8
FreeMemory 51
FreeMemory (discontinued) 8

- G -

GenFileNames 22
GetCol 22, 51
GetColBase 22
GetColIPSolution 22
GetColLB 22
GetColLPSolution 22
GetColType 22
GetColUB 22
GetCost 22
GetDim 22, 51
GetIPObjValue 22
GetIPSolution 8, 51
GetLPObjValue 22
GetLPSolution 8, 51
GetMaxDim 22, 51
GetModel 8, 51
GetNonzero 22, 51
GetNumberOfColumns 22
GetNumberOfNZ 22
GetNumberOfRows 22
GetParameter 8, 51
GetRedCost 22
GetRow 8, 51
GetRowBase 22
GetRowDualValues 22
GetRowIPSolution 22
GetRowLhs 22
GetRowLPSolution 22
GetRowRhs 22
GetRowType 22
GetSolutionStatus 22

- I -

Initialize 8
InitModel 8, 51
Interlanguage Interface 54

- L -

LIB 51
LoadModel 8, 51

- M -

Main Window 58
Mops 8, 51
MOPS.LIB 51

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

MPS File 145
MPS file example 126
MPS Files 125

- O -

Optimize 8, 51
OptimizeIP 51
OptimizeIP (discontinued) 8
OptimizeLP 51
OptimizeLP (discontinued) 8

- P -

PutCol 22, 51
PutModel 8, 51
PutNonzeros 22, 51
PutRow 22, 51

- R -

ReadMpsFile 8, 51
ReadProfile 8, 51
ReadTripletFile 8, 51
Ressource Limits 79

- S -

SetMaxIPDim 22, 51
SetMaxLPDim 22, 51
SetParameter 8, 51
Static Library 51

- T -

Triplet File 146

- V -

Visual Basic .NET 44
Visual Basic 6.0 43
Visual C/C++ 6.0 46

- W -

WriteMpsFile 8, 51
WriteTripletFile 8, 51

- X -

xabgap 79
xadcol 79
xadnon 79
xadrow 79
xallocm 51
xbatyp 69
xbndha 69
xbound 77
xchksw 89
xcmgap 113
xctime 113
xdata 77
xdecol 51
xderow 51
xdimcn 89
xdjscl 69
xdropf 69
xdropm 69, 84
xertyp 113
xfiltl 69
xfname 51
xfnbai 66
xfnbao 66
xfnips 66
xfnlog 66
xfnlps 66
xfnmps 66
xfnmsg 66
xfnpro 66
xfnsav 66
xfnsta 66
xfrbas 69
xfreem 51
xfrinv 69
xfrlog 89
xfrnod 95
xglgap 79
xgtcol 51
xgtips 51
xgtlps 51
xgtmod 51
xgtnz 51

© 2007 MOPS Optimierungssysteme GmbH & Co. KG

xgtrow 51
ximrrw 89
xinf 89
xinfor 89
xinitm 51
xipdua 69
xipfun 113
xipmem 69
xipsta 113
xiptim 113
xipxov 69
xiter 113
xline1 113
xline2 113
xlpfun 113
xlpsta 113
xlptim 113
xlptyp 69
xluctr 69
xm 113
xmaxel 89
xminmx 89
xmiter 79
xmitip 79
xmmax 79
xmorig 113
xmreal 79
xmxdsk 79
xmxj 113
xmxj1 113
xmxmin 79
xmxnod 79
xmxtlp 79
xn 113
xnenmx 79
xnif 113
xnints 113
xnmax 79
xnodes 113
xnproc 69
xnwcon 89
xnwluf 69
xnwluu 69
xnzero 113
xnzmax 79
xobj 77
xoptim 51
xoptip 51
xoptlp 51
xoripm 69
xoutlv 89
xoutsl 89
xpreme 69
xptcol 51
xptmod 51
xptnzs 51
xptrow 51
xrange 77
xrcmax 89
xrdmps 51
xrdtri 51
xrduce 69
xrhs 77
xrimpr 84
xrprof 51
xrtcod 113
xscale 89
xsif 113
xstart 69
xstmps 51
xstorn 89
xsttri 51
xtold1 84
xtold2 84
xtolin 84
xtolpv 84
xtolqi 84
xtolr1 84
xtolr2 84
xtolre 84
xtolx 84
xtolx1 84
xtolx2 84
xtolzr 84
xversn 113
xwrmps 89
xzbest 113
xzlbnd 113
xzubnd 113

© 2007 MOPS Optimierungssysteme GmbH & Co. KG